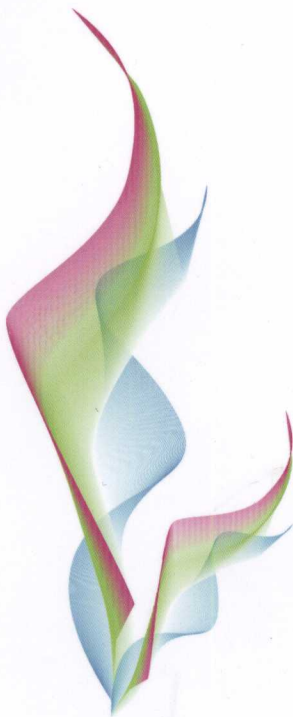


## 版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。





The Practice of Machine Learning, Second Edition

# 机器学习实践指南

## 案例应用解析

第2版

麦好◎著



机械工业出版社  
China Machine Press

最近几年，借着大数据的东风，机器学习一直保持着强劲的发展势头，资本市场也一度为之“疯狂”，到了2015年，人类在机器学习领域取得了一系列重大突破，2016年及未来若干年将是“机器学习”的发力阶段。本书致力于阐述机器学习实践的必备知识，第1版发行以后，深受广大读者的喜爱。然而，由于机器学习涉及的内容繁多，以致必须了解的知识也很庞杂，而第1版成书匆忙，部分内容缺失或描述不够细致，因此第2版在此基础上进行了完善，增补内容达40%以上。本书分为四部分，由浅入深讲解机器学习算法，希望能帮助更多的朋友进入机器学习的精彩世界。本书的读者QQ群为192029861。

## 作者简介

---

**麦好** 计算机专业工程硕士，CSDN专家，青年海归协会会员，中国量化投资学会山西分会成员，目前从事金融智能算法与投资数据分析工作。有十余年架构设计及算法设计经验，先后就职于多家软件科技公司、电子科技公司，实战经验丰富，擅长使用C、C++、Python、汇编、R、SAS等语言，参与过信息系统核心组件、社区插件、垂直搜索引擎、文本分析系统、通信信息隐藏、视频服务、汇编底层设计、基于分布式计算平台的网络数据爬取与分析等项目。近期关注计算神经学、分子生物学、智能机器人、物联网等方向。

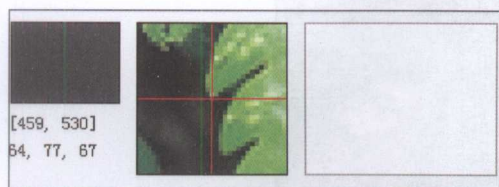


图 3-6 树叶放大的颗粒效果



图 3-9 随机产生若干像素点



图 3-10 图像变暗



图 3-11 图像变亮



图 3-12 图像日落效果



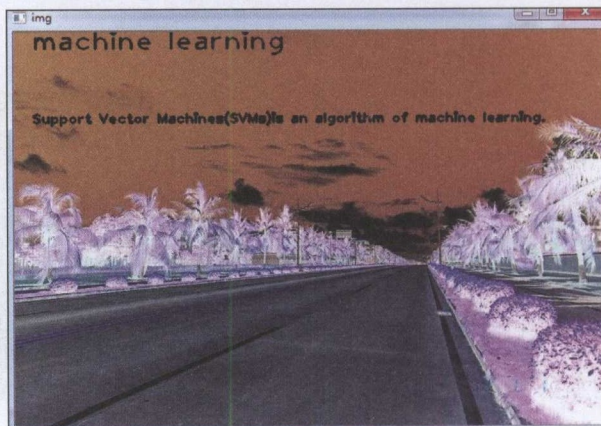


图 3-13 负片和水印效果

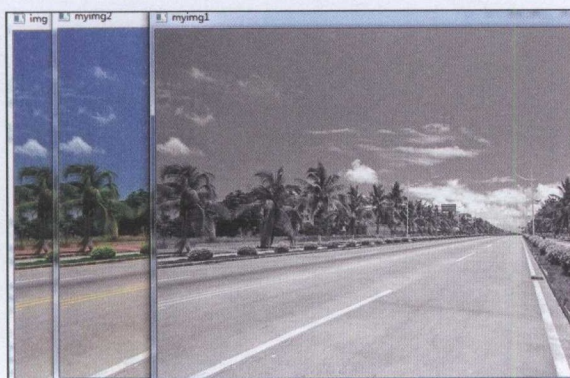


图 3-18 图像灰度化

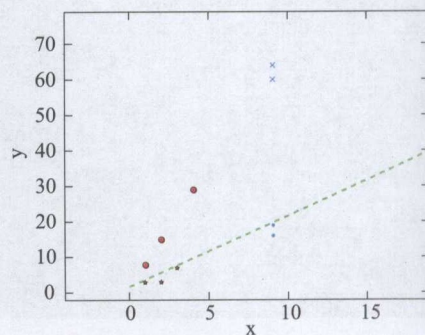


图 8-4 神经网络分类

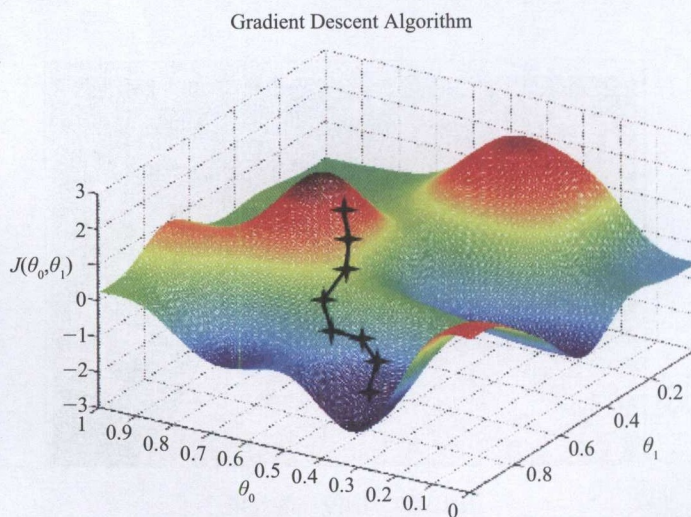


图 8-6 误差曲面及梯度下降



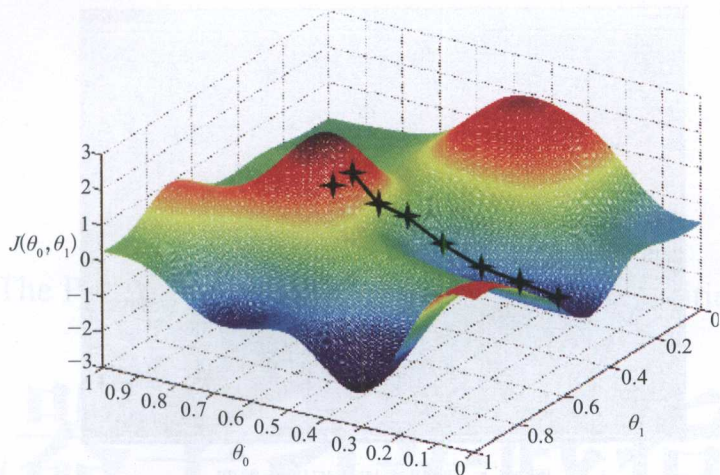


图 8-7 落到局部最小点的梯度下降

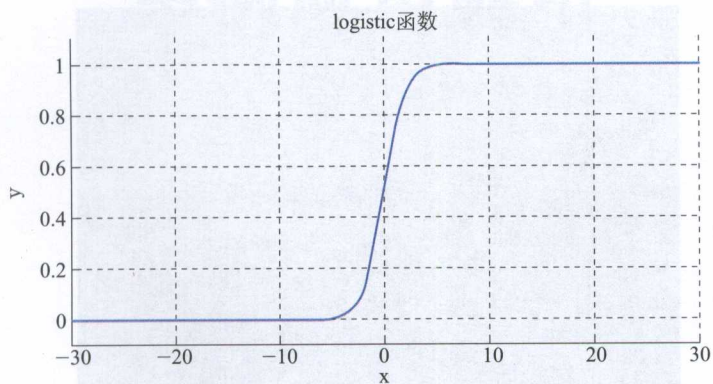


图 8-12 sigmoid 曲线

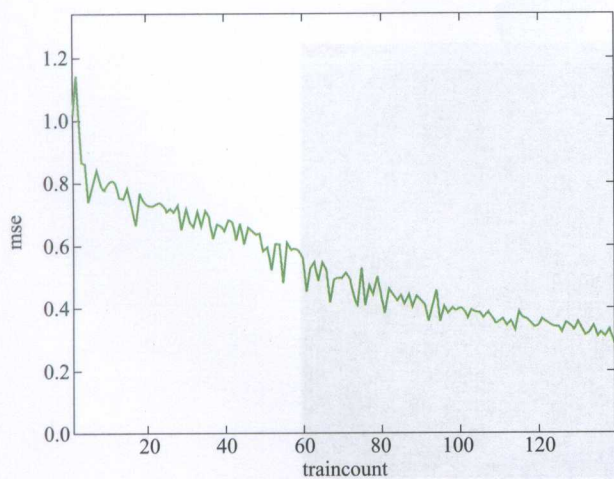


图 8-18 误差曲线

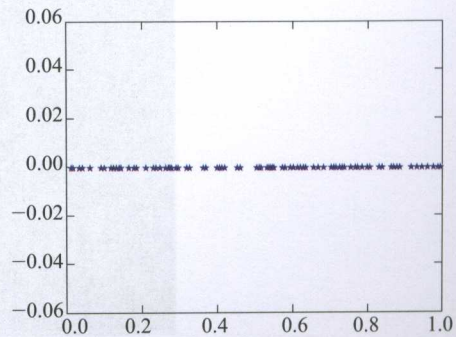


图 8-23 数据点分布

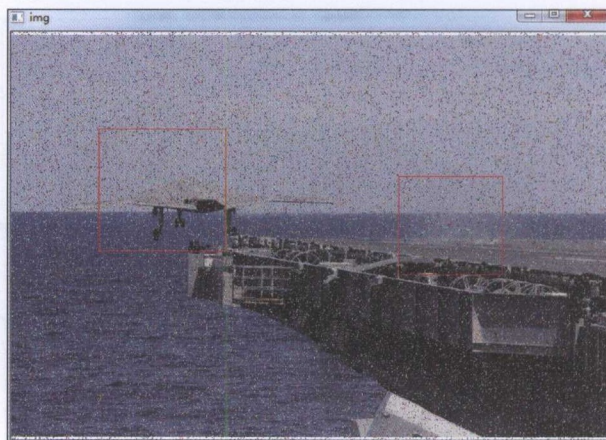


图 10-5 弱噪声切片识别效果图



图 10-6 强噪声图像



图 10-7 强噪声切片识别效果图





技术丛书

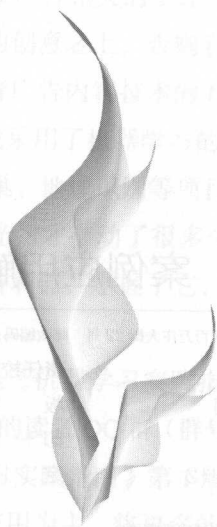
The Practice of Machine Learning, Second Edition

# 机器学习实践指南

## 案例应用解析

第2版

麦好◎著



机械工业出版社  
China Machine Press

## 图书在版编目 (CIP) 数据

机器学习实践指南: 案例应用解析 / 麦好著. —2 版. —北京: 机械工业出版社, 2016.7  
(大数据技术丛书)

ISBN 978-7-111-54021-2

I. 机… II. 麦… III. 机器学习-指南 IV. TP181-62

中国版本图书馆 CIP 数据核字 (2016) 第 130361 号

## 机器学习实践指南: 案例应用解析 (第 2 版)

出版发行: 机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码: 100037)

责任编辑: 余 洁

责任校对: 殷 虹

印 刷: 北京诚信伟业印刷有限公司

版 次: 2016 年 7 月第 2 版第 1 次印刷

开 本: 186mm × 240mm 1/16

印 张: 34 (含 0.25 印张彩插)

书 号: ISBN 978-7-111-54021-2

定 价: 89.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88379426 88361066

投稿热线: (010) 88379604

购书热线: (010) 68326294 88379649 68995259

读者信箱: hzit@hzbook.com

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问: 北京大成律师事务所 韩光 / 邹晓东



## Foreword 推荐序

在 2013 年的中国大数据创新峰会上，我偶然结识了作者，期间聊到了人工智能革命和机器学习的话题，被作者的渊博知识所折服，慢慢地结下了深厚的友谊。时间一晃而过，机器学习现在已经得到了井喷式发展，按照麻省理工学院罗德尼·布鲁克斯的预测：到 2100 年以前，我们的日常生活中将充满智能机器人，而且人类无法将自己同它们区分开来，我们也将是机器人，同机器人互相联系。

追忆 2011 年，当时我在吉林大学读研三，幸运地拿到了百度研发工程师的 offer，进入百度商务搜索架构部，一直做着与凤巢广告相关的工作。现代广告业的奠基人大卫·奥格威曾经说过，除非你的广告建立在伟大的创意之上，否则它就像夜航的船，不为人所注意。广告的创意是广告的灵魂，我也一直沿着广告内容技术的方向，优化创意，提升用户的体验，提升广告主的转化。在这个方向上，我采用了机器学习的相关技术，取得了毕昇（获得 2014 年度百度最高奖）、图片凤巢、知识凤巢、地域识别等项目的成功，深刻地体会到了机器学习的强大，正是有了机器学习的闪闪发光，才推动了很多令人惊艳的产品的诞生。对于互联网、IT 从业人员，机器学习已经成为必备利器，掌握了它，就等于站在了巨人的肩膀上工作，可帮助自己提高个人的核心竞争力。

我和作者认识近 3 年，同时也是《机器学习实践指南》第 1 版的读者，并在工作之余与作者一起管理《机器学习实践指南》的读者 QQ 群（群号：192029861），在群里认识了更多专注机器学习的朋友和学者。《机器学习实践指南》第 1 版主要针对初、中级读者，作者出书的目标就是：以机器学习算法的实践应用为主，将更多的“门外汉”带入机器学习殿堂，让更多拥有机器学习理论却无法下手的朋友掌握机器学习实践思维，轻松步入机器学习实战领域。实践思维对 IT 行业非常重要，一旦形成了适当的思维方式，很多工作中遇到的技术难题将迎刃而解，学习新知识的速度也更快，因为只有实践与理论相结合才能更精准地理解知识。也希望对机器学习有兴趣的读者能从中受益。

## 图书在版编目(CIP)数据

《机器学习实践指南》第2版出版在即,我高兴地接受了作者的邀请——为本书写推荐序。第2版比第1版增加了更多的案例和算法解析,全书详细介绍了机器学习发展及应用前景、科学计算平台、Python 计算平台应用、R 语言计算平台应用、生产环境基础、统计分析基础、描述性分析案例、假设检验与回归模型案例、神经网络、统计算法、欧氏距离与余弦相似度、SVM、回归算法、PCA 降维、关联规则、聚类与分类算法、数据拟合案例、图像算法案例、机器视觉案例、文本分类案例等机器学习实践与应用。

第2版致力推动机器学习理论在国内的普及和应用,为公司创建更多的商业价值;同时,力争让更多的学生、IT 工程师等进入人工智能相关领域,适应智能时代工作的需要。

最后,希望大家喜欢这本书,进而从中受益。

徐培治

百度在线网络技术(北京)有限公司

2016年3月于北京

## Preface 前言

### 为什么要写这本书

随着全球第三次工业革命的迅猛发展，机器学习技术异军突起，人类对机器学习技术的研究也开辟出了许多全新的应用领域，这使智能机器的计算能力和可定制性上升到了一个新的层次。到了 2015 年，人类在机器学习领域取得了一系列重大的突破，这项技术已悄无声息地潜入我们的日常生活，而在未来，机器学习也将拥抱变化，持续发力。如今，它已经在各行各业的技术革新中扮演着日益重要的角色，从各方面影响和改变着我们的生活。

近年来，机器学习技术在国外得到了海量应用和深入发展。2015 年 11 月，谷歌开源了全新的 TensorFlow 机器学习系统，该系统更快、更智能，也更具有弹性。2015 年 1 月，机器学习平台 GraphLab 改名为 Dato，并获得了 1850 万美元的新融资（投资方为 Vulcan Capital、Opus Capital、New Enterprise Associates、Madrona Venture Group），此前他们曾获得 680 万美元的融资。2015 年 8 月，Facebook 推出了“M”，Facebook 认为人类不仅会回答人工智能所不能回答的问题，而且从长远来看，人类也会帮助改善人工智能技术，“M”除了能做到回答问题、查阅信息等基本功能外，还可以帮助用户完成如购买商品、餐厅定位、安排旅行计划等操作。在 2015 年 12 月召开的“2015 年神经信息处理系统”（NIPS）会议上，微软研究人员和工程师公开了 20 多篇机器学习最新研究成果的论文。此外，微软还宣布，机器学习正在成为 Windows 10 的一部分：Skype 翻译可以将口语几乎实时地翻译成其他语言，就像《星际迷航》中的通用翻译器那样，可以做到面对面的交流。Cortana 个人数字助理在与用户的互动中不断学习与改进，从而帮助用户管理日历、跟踪快递，甚至能与用户聊天和讲笑话，实现真正的个性化互动体验。Clutter 是微软 Office 2016 的成员，通过学习它可以识别出哪些电子邮件对用户来说最重要，并自动将不重要的邮件重定向到一个单独的文件夹中，从而保持用户收件箱的整洁。2015 年 9 月，美军军队医疗中心指挥官少将 Steve Jones 在美军陆军的一次

会议上发言表示,未来可以让智能机器人代替人类上战场运送伤员,美国军方甚至高调宣布:未来战场上机器人救起的可能不是人,而是机器人,因为智能机器人军团将代替人类出征。

在国内,机器学习掀起了技术革新的热潮,智能技术得到了广泛的普及和应用。隶属于中国科学院的新松机器人自动化公司生产了智能复合型机器人,这个安装了眼睛和感知器件的智能机器人,可以在车间里自由地行走并十分精确地完成任务,当其他工位人手不足时,接到指令的他还会主动上前帮忙,马上进入角色并开始工作。百度创造和完善了大规模机器学习的技术,搭建了一个能容纳万亿特征数据的、分钟级别模型更新的、高效训练的点击率预估系统;为进一步深入地发展机器学习技术,百度开始研究如何从“机器学习”到“复制人类大脑”;此外,百度甚至在2016年提出,百度的产品和服务都靠机器学习等技术来驱动。

随着机器学习技术在国内外的应用,机器学习工程师成为炙手可热的职位。现在中国已经悄然兴起了机器学习的学习热潮,掌握了机器学习技术的工程师将成为各大IT巨头疯抢的“香馍馍”,良好的发展势头和较高的职业薪水,吸引着越来越多的软件工程师和数据分析师涌入机器学习的领域。国内知名的公司百度、阿里巴巴、腾讯(俗称BAT)为迎接大数据时代带来的挑战,早已全面引进机器学习方面的人才,并有组织地对机器学习技术展开大规模的、更深入的研究。其他各大公司(包括非IT行业的公司)也提出了引进机器学习研发工程师的渴求。

但是,机器学习的入门门槛较高,尤其是对研究者的数学理解能力有较高的要求,相对于数据结构、算法导论中讲述的计算机算法及系统架构知识来说,机器学习是一个全新的领域,理解机器学习算法往往要从理解它所涉及的数学公式和数学知识开始,打好数学基础是非常有必要的,一旦掌握了数学分析、线性代数、概率与统计、统计学、离散数学、抽象代数、数学建模等数学理论后,理解机器学习算法就会容易很多,不再畏惧那些让人生厌的、麻烦的数学符号和数学公式,说不定还会喜欢上这些数学公式,并亲自推导一番。希望本书能帮助朋友们进入机器学习的精彩世界。

## 读者对象

- 开发人员。在理解机器学习算法的基础上,调用机器学习的中间库进行开发,将机器学习应用于各种场景,如数据分析、图像识别、文本分类、搜索引擎、中文智能输入法等。
- 架构师。在理解机器学习算法的基础上,适应现代云计算平台的发展,将机器学习算法应用在大规模的并行计算上。同时,机器学习算法是大数据分析的基础,如神经网络、SVM、相似度分析、统计分析等技术。



□ 机器学习的初、中级读者。人类对机器学习的研究只是一个开始，还远远没有结束。

近年来，机器学习一直保持着强劲的发展势头，并拥有美好的发展前景，这点不同于某些软件开发领域中的程序语言或架构知识。掌握机器学习技术有一定的难度，但也意味着，掌握机器学习的技术就能获得更高的薪水和更具前景的职业。

## 如何阅读本书

全书分为准备篇、基础篇、统计分析实战篇和机器学习实战篇。机器学习算法建立在复杂的计算理论基础之上，并涉及多门数学学科。抽象的理论加上成堆的数学公式，给部分读者带来了极大的挑战，将渴求学习的人们挡在了门外。针对这种情况，本书力求理论联系实际，在介绍理论基础的同时，注重机器学习算法的实际运用，让读者更好地明白其中的原理。

准备篇中首先将介绍机器学习的发展及应用前景，使读者产生浓厚的兴趣，同时也将介绍目前常用的科学计算平台和本书将用到的工程计算平台，使读者消除对机器学习的畏难情绪，这些平台的使用也降低了机器学习软件实现的难度。

基础篇将介绍数学知识基础和计算平台应用实例，介绍计算平台的开发基本知识，并应用这些平台实现计算应用。

最后，本书将针对统计分析实战和机器学习实战两个部分帮助读者建立机器学习实战指南，应用计算平台对统计分析及机器学习算法进行实现和应用，同时还会附上效果图，让读者对机器学习的基本应用和理论基础有一个形象的理解。

## 勘误和支持

由于作者的水平有限，编写的时间也很仓促，书中难免会出现一些错误或不准确的地方，不妥之处恳请读者批评指正。如果遇到任何问题，或有更多的宝贵意见，欢迎发送邮件至我的邮箱 [myhaspl@myhaspl.com](mailto:myhaspl@myhaspl.com)，很期待能够听到您的真挚反馈。此外，本书的代码及相关资源（包括思考题中涉及的数据等）的下载地址为：<https://yunpan.cn/cYjhBYGLKkKTb>（提取码：65ad）。

## 致谢

首先我要感谢伟大的电影《机械公敌》及其主角威尔·史密斯，这位美国演员主演了《当幸福来敲门》《拳王阿里》《绝地战警》《全民超人汉考克》《黑衣人》《机械公敌》，他曾获奥斯卡奖和金球奖提名。他主演的《当幸福来敲门》让很多人理解到了幸福是什么，而《机械公敌》让我看到了人工智能的未来，我相信《机械公敌》描述的以下场景在将来一定能

实现:

公元 2035 年, 智能型机器人已被人类广泛利用, 作为最好的生产工具和人类伙伴, 机器人在各个领域扮演着日益重要的角色。而由于众所周知的机器人“三大安全法则”的限制, 人类对这些能够胜任各种工作且毫无怨言的伙伴充满信任, 它们中的很多甚至已经成为各个家庭的组成成员。

在此, 我衷心地感谢机械工业出版社华章公司的编辑杨福川老师和策划编辑杨绣国老师, 由于他们的魄力和远见, 让我顺利地完成了全部书稿。最后我要感谢家人的大力支持和无私奉献, 正因为有他们的关心和照顾, 我才有足够的时间和精力来完成本书的撰写工作。

谨以此书, 献给热爱机器学习的朋友, 以及喜欢威尔·史密斯的影迷。

麦好 (Myhaspl)

2016 年 3 月于中国广东

# 目 录 Contents

推荐序

前言

## 第一部分 准备篇

### 第1章 机器学习发展及应用前景 ..... 2

#### 1.1 机器学习概述 ..... 2

##### 1.1.1 什么是机器学习 ..... 3

##### 1.1.2 机器学习的发展 ..... 3

##### 1.1.3 机器学习的未来 ..... 4

#### 1.2 机器学习应用前景 ..... 5

##### 1.2.1 数据分析与挖掘 ..... 5

##### 1.2.2 模式识别 ..... 6

##### 1.2.3 更广阔的领域 ..... 6

#### 1.3 小结 ..... 7

### 第2章 科学计算平台 ..... 8

#### 2.1 科学计算软件平台概述 ..... 9

##### 2.1.1 常用的科学计算软件 ..... 9

##### 2.1.2 本书使用的工程计算平台 ..... 10

#### 2.2 计算平台的配置 ..... 11

##### 2.2.1 Numpy 等 Python 科学计算包的 安装与配置 ..... 11

##### 2.2.2 OpenCV 安装与配置 ..... 14

##### 2.2.3 mlp 安装与配置 ..... 14

##### 2.2.4 BeautifulSoup 安装与配置 ..... 15

##### 2.2.5 Neurolab 安装与配置 ..... 15

##### 2.2.6 R 安装与配置 ..... 16

#### 2.3 小结 ..... 16

## 第二部分 基础篇

### 第3章 计算平台应用实例 ..... 18

#### 3.1 Python 计算平台简介及 应用实例 ..... 18

##### 3.1.1 Python 语言基础 ..... 18

##### 3.1.2 Numpy 库 ..... 29

##### 3.1.3 pylab、matplotlib 绘图 ..... 36

##### 3.1.4 图像基础 ..... 38

##### 3.1.5 图像融合与图像镜像 ..... 46

##### 3.1.6 图像灰度化与图像加噪 ..... 48

##### 3.1.7 声音基础 ..... 51

##### 3.1.8 声音音量调节 ..... 53

##### 3.1.9 图像信息隐藏 ..... 58

##### 3.1.10 声音信息隐藏 ..... 62

#### 3.2 R 语言基础 ..... 68

3.2.1 基本操作 .....	69	4.4.1 Citrix Xenserver 概述 .....	125
3.2.2 向量 .....	71	4.4.2 Citrix Xenserver 部署 .....	126
3.2.3 对象集属性 .....	77	4.4.3 基于 XenCenter 的虚拟 服务器管理 .....	126
3.2.4 因子和有序因子 .....	78	4.5 Linux 环境下的 NumPy 安装 .....	135
3.2.5 循环语句 .....	79	4.6 Linux 环境下的 R 运行环境 .....	136
3.2.6 条件语句 .....	79	4.7 PyPy 编译器 .....	136
3.3 R 语言科学计算 .....	80	4.7.1 PyPy 概述 .....	136
3.3.1 分类(组)统计 .....	80	4.7.2 PyPy 安装与配置 .....	137
3.3.2 数组与矩阵基础 .....	81	4.7.3 PyPy 性能 .....	137
3.3.3 数组运算 .....	84	4.7.4 PyPy 实践之 Lempel-Ziv 压缩 .....	138
3.3.4 矩阵运算 .....	85	4.8 小结 .....	145
3.4 R 语言计算实例 .....	93	思考题 .....	146
3.4.1 学生数据集读写 .....	93		
3.4.2 最小二乘法拟合 .....	94		
3.4.3 交叉因子频率分析 .....	96		
3.4.4 向量模长计算 .....	97		
3.4.5 欧氏距离计算 .....	98		
3.5 小结 .....	99		
思考题 .....	99		
<b>第4章 生产环境基础</b> .....	<b>100</b>		
4.1 Windows Server 2008 基础 .....	100		
4.1.1 Windows Server 2008 R2 概述 .....	101		
4.1.2 Windows PowerShell .....	102		
4.2 Linux 基础 .....	103		
4.2.1 Linux 命令 .....	104		
4.2.2 Shell 基础 .....	114		
4.3 Vim 编辑器 .....	122		
4.3.1 Vim 编辑器概述 .....	122		
4.3.2 Vim 常用命令 .....	123		
4.4 虚拟化平台 .....	124		
		<b>第三部分 统计分析实战篇</b>	
		<b>第5章 统计分析基础</b> .....	<b>148</b>
		5.1 数据分析概述 .....	148
		5.2 数学基础 .....	149
		5.3 回归分析 .....	154
		5.3.1 单变量线性回归 .....	154
		5.3.2 多元线性回归 .....	156
		5.3.3 非线性回归 .....	157
		5.4 数据分析基础 .....	159
		5.4.1 区间频率分布 .....	159
		5.4.2 数据直方图 .....	161
		5.4.3 数据散点图 .....	162
		5.4.4 五分位数 .....	164
		5.4.5 累积分布函数 .....	165
		5.4.6 核密度估计 .....	166
		5.5 数据分布分析 .....	167



5.6 小结 .....	169
思考题 .....	170

## 第6章 描述性分析案例 .....

6.1 数据图形化案例解析 .....	171
6.1.1 点图 .....	171
6.1.2 饼图和条形图 .....	172
6.1.3 茎叶图和箱线图 .....	173
6.2 数据分布趋势案例解析 .....	175
6.2.1 平均值 .....	175
6.2.2 加权平均值 .....	175
6.2.3 数据排序 .....	176
6.2.4 中位数 .....	177
6.2.5 极差、半极差 .....	177
6.2.6 方差 .....	178
6.2.7 标准差 .....	178
6.2.8 变异系数、样本平方和 .....	178
6.2.9 偏度系数、峰度系数 .....	179
6.3 正态分布案例解析 .....	180
6.3.1 正态分布函数 .....	180
6.3.2 峰度系数分析 .....	181
6.3.3 累积分布概率 .....	181
6.3.4 概率密度函数 .....	182
6.3.5 分位点 .....	183
6.3.6 频率直方图 .....	185
6.3.7 核概率密度与正态概率 分布图 .....	185
6.3.8 正态检验与分布拟合 .....	186
6.3.9 其他分布及其拟合 .....	188
6.4 多变量分析 .....	189
6.4.1 多变量数据分析 .....	189
6.4.2 多元数据相关性分析 .....	197

6.5 小结 .....	201
思考题 .....	201

## 第7章 假设检验与回归模型案例 .....

7.1 假设检验 .....	202
7.1.1 二项分布假设检验 .....	202
7.1.2 数据分布检验 .....	204
7.1.3 正态总体均值检验 .....	205
7.1.4 列联表 .....	206
7.1.5 符号检测 .....	207
7.1.6 秩相关检验 .....	210
7.1.7 Kendall 相关检验 .....	213
7.2 回归模型 .....	214
7.2.1 回归预测与显著性检验 .....	214
7.2.2 回归诊断 .....	216
7.2.3 回归优化 .....	217
7.2.4 主成分回归 .....	219
7.2.5 广义线性模型 .....	221
7.3 小结 .....	226
思考题 .....	226

## 第四部分 机器学习实战篇

## 第8章 机器学习算法 .....

8.1 神经网络 .....	230
8.1.1 Rosenblatt 感知器 .....	232
8.1.2 梯度下降 .....	245
8.1.3 反向传播与多层感知器 .....	251
8.1.4 Python 神经网络库 .....	270
8.2 统计算法 .....	272
8.2.1 平均值 .....	272
8.2.2 方差与标准差 .....	274

8.2.3 贝叶斯算法	276	9.1.2 神经网络拟合法	338
8.3 欧氏距离	279	9.2 线性滤波	352
8.4 余弦相似度	280	9.2.1 WAV 声音文件	352
8.5 SVM	281	9.2.2 线性滤波算法过程	352
8.5.1 数学原理	281	9.2.3 滤波 Python 实现	353
8.5.2 SMO 算法	283	9.3 数据或曲线平滑	358
8.5.3 算法应用	283	9.3.1 平滑概述	358
8.6 回归算法	287	9.3.2 移动平均	359
8.6.1 线性代数基础	288	9.3.3 递归线性过滤	362
8.6.2 最小二乘法原理	289	9.3.4 指数平滑	364
8.6.3 线性回归	290	9.4 小结	368
8.6.4 多元非线性回归	292	思考题	368
8.6.5 岭回归方法	294		
8.6.6 伪逆方法	295	<b>第 10 章 图像算法案例</b>	<b>370</b>
8.7 PCA 降维	296	10.1 图像边缘算法	370
8.8 关联规则	297	10.1.1 数字图像基础	370
8.8.1 关联规则概述	297	10.1.2 算法描述	371
8.8.2 频繁项集算法	298	10.2 图像匹配	372
8.8.3 关联规则生成	301	10.2.1 差分矩阵求和	373
8.8.4 实例分析	302	10.2.2 差分矩阵均值	375
8.9 自动分类	306	10.2.3 欧氏距离匹配	376
8.9.1 聚类算法	306	10.3 图像分类	382
8.9.2 决策树	313	10.3.1 余弦相似度	382
8.9.3 AdaBoost	316	10.3.2 PCA 图像特征提取算法	388
8.9.4 竞争型神经网络	317	10.3.3 基于神经网络的图像 分类	389
8.9.5 Hamming 神经网络	323	10.3.4 基于 SVM 的图像分类	394
8.10 小结	325	10.4 高斯噪声生成	397
思考题	325	10.5 二值化	401
<b>第 9 章 数据拟合案例</b>	<b>327</b>	10.5.1 threshold	401
9.1 数据拟合	327	10.5.2 adaptiveThreshold	402
9.1.1 图像分析法	327	10.6 插值与缩放	404

10.7 仿射 .....	405	11.3.2 差分算法 .....	452
10.7.1 仿射原理 .....	405	11.3.3 光流法 .....	456
10.7.2 仿射变换实例 .....	405	11.4 形状检测 .....	458
10.8 透视投影与透视变换 .....	406	11.4.1 KNN 算法概述 .....	458
10.8.1 透视投影原理 .....	406	11.4.2 形状特征提取 .....	459
10.8.2 透视投影实例 .....	407	11.4.3 形状分类 .....	459
10.9 灰度变换与图像增强 .....	409	11.5 小结 .....	462
10.9.1 灰度变换概述 .....	409	思考题 .....	462
10.9.2 对数变换 .....	409		
10.9.3 分段线性变换 .....	410	<b>第 12 章 文本分类案例</b> .....	463
10.9.4 指数变换 .....	411	12.1 文本分类概述 .....	463
10.9.5 直方图均衡化 .....	412	12.2 余弦相似度分类 .....	464
10.10 图像滤波与除噪 .....	415	12.2.1 中文分词 .....	465
10.10.1 均一化块滤波 .....	415	12.2.2 停用词清理 .....	467
10.10.2 邻域平均法 .....	420	12.2.3 算法实战 .....	468
10.10.3 中值滤波 .....	423	12.3 朴素贝叶斯分类 .....	473
10.10.4 高斯滤波 .....	427	12.3.1 算法描述 .....	473
10.10.5 双边滤波 .....	429	12.3.2 先验概率计算 .....	474
10.10.6 卷积滤波 .....	431	12.3.3 最大后验概率 .....	474
10.10.7 边缘检测 .....	433	12.3.4 算法实现 .....	474
10.11 小结 .....	435	12.4 自然语言处理 .....	480
思考题 .....	435	12.4.1 NLTK 简介 .....	480
<b>第 11 章 机器视觉案例</b> .....	437	12.4.2 NLTK 与 jieba 的配置 .....	481
11.1 人脸辨识 .....	437	12.4.3 中文分词并标注词性 .....	483
11.1.1 人脸定位 .....	437	12.4.4 词特征指标分析 .....	484
11.1.2 人脸辨识 .....	439	12.4.5 Web 文档分析 .....	499
11.2 手写数字识别 .....	446	12.4.6 Web 文档的朴素贝叶斯 分类 .....	503
11.2.1 手写数字识别算法 .....	446	12.4.7 语法结构分析 .....	515
11.2.2 算法的 Python 实现 .....	447	12.4.8 Web 文档聚类 .....	518
11.3 运动侦测 .....	449	12.5 小结 .....	526
11.3.1 视频采集 .....	450	思考题 .....	526

# 第一部分 Part 1

## 第一部分 *Part 1*

# 准备篇

# 准备篇

子贡问为仁。子曰：“工欲善其事，必先利其器。居是邦也，事其大夫之贤者，友其士之仁者。”

——孔子

# 机器学习发展及应用前景

纵观国内软件工程师的发展路线，前期多以程序员（“码农”）、测试工程师、数据库管理员、多媒体技术员、网页与信息技术员等职业为主；中期主要是软件设计师、软件评测师、技术支持师等职业；后期是职业发展的黄金阶段，这一阶段对于拥有丰富技术经验的工程师来说十分重要，但这一阶段容易遭遇到技术发展瓶颈，因此，很多人将目光投向了项目管理和系统架构，比如：系统架构师、项目管理师等。

近年来，国内对机器学习的研究日益深入，应用领域不断扩大，催生了新的 IT 职位——机器学习工程师，百度、搜狗、阿里巴巴、淘宝、奇虎等国内 IT 巨头纷纷提出了对机器学习工程师的需求，掌握机器学习的人才成为了各大 IT 厂商争抢的“香饽饽”。机器学习迅速走红成为热门技术，这给软件工程师带来了绝佳的发展机遇，研究与应用机器学习算法成为了突破技术瓶颈的方式，机器学习工程师、项目管理师和系统架构师并称为后期发展的三大黄金职位。

## 1.1 机器学习概述

机器学习作为一门多领域的交叉学科，在近 20 年里异军突出。机器学习涉及概率论、统计学、微积分、代数学、算法复杂度理论等多门学科。通过可以让计算机自动“学习”的算法来实现人工智能，是人类在人工智能领域展开的积极探索。

2009 年，被誉为人工大脑之父的雨果·德·加里斯教授走进清华大学讲堂，在两小时的演讲时间内，给大家描述了一个人工智能的世界：20 年后，人工智能机器可以和人类做朋友，50 年后，人工智能将成为人类最大的威胁，世界最终会因人工智能超过人类而爆发一场战争，这场智能战争也许会夺去数十亿人的生命。这样的描述并不是幻想，随着人类在人工



智能领域取得的进步,这很有可能成为事实。而这一切主要归功于对机器学习的研究和探索。

### 1.1.1 什么是机器学习

学习是人类具有的一种重要智能行为。人类一直梦想机器能像人类一样学习,也一直在为这个终极目标努力。那么,什么是机器学习呢?长期以来众说纷纭,Langley (1996) 定义机器学习为:“机器学习是一门人工智能的科学,该领域的主要研究对象是人工智能,特别是如何在经验学习中改善具体算法的性能”(Machine learning is a science of the artificial. The field's main objects of study are artifacts, specifically algorithms that improve their performance with experience.). Mitchell (1997) 在《Machine Learning》中写道:“机器学习是计算机算法的研究,并通过经验提高其自动进行改善”(Machine Learning is the study of computer algorithms that improve automatically through experience.). Alpaydin (2004) 提出自己对机器学习的定义:“机器学习是用数据或以往的经验,来优化计算机程序的性能标准”(Machine learning is programming computers to optimize a performance criterion using example data or past experience.).

笔者综合维基百科和百度百科的定义,尝试着将机器学习定义如下:“机器学习是一门人工智能的科学,该领域的主要研究对象是人工智能,专门研究计算机怎样模拟或实现人类的学习行为,以获取新的知识或技能,重新组织已有的知识结构使之不断改善自身的性能,它是人工智能的核心,是使计算机具有智能的根本途径。机器学习的研究方法通常是根据生理学、认知科学等对人类学习机理的了解,建立人类学习过程的计算模型或认识模型,发展各种学习理论和学习方法,研究通用的学习算法并进行理论上的分析,建立面向任务的具有特定应用的学习系统。”

### 1.1.2 机器学习的发展

早在古代,人类就萌生了制造出智能机器的想法。中国人在4500年前发明的指南车,以及三国时期诸葛亮发明的尽人皆知的木牛流马;日本人在几百年前制造过靠机械装置驱动的玩偶;1770年英国公使给中国皇帝进贡了一个能写“八方向化,九土来王”8个汉字的机器玩偶(这个机器人至今还保存在故宫博物院),等等。这些例子,都只是人类早期对机器学习的一种认识和尝试。

真正的机器学习研究起步较晚,它的发展过程大体上可分为以下4个时期:

第一阶段是在20世纪50年代中叶到20世纪60年代中叶,属于热烈时期。

第二阶段是在20世纪60年代中叶至20世纪70年代中叶,被称为机器学习冷静期。

第三阶段是从20世纪70年代中叶至20世纪80年代中叶,称为机器学习复兴期。

最新的阶段起始于1986年。当时,机器学习综合应用了心理学、生物学和神经生理学以及数学、自动化和计算机科学,并形成了机器学习理论基础,同时还结合各种学习方法取长补短,形成集成学习系统。此外,机器学习与人工智能各种基础问题的统一性观点正在形成,各种学习方法的应用范围不断扩大,同时出现了商业化的机器学习产品,还积极开展了

与机器学习有关的学术活动。

近年来,随着全球第三次工业革命的迅猛发展,机器学习在国内外得到了广泛的应用和发展,如今,它已经在各行各业的技术革新中扮演着日益重要的角色,从各方面影响和改变着我们的生活。

2015年8月,Facebook推出了“M”,Facebook认为人类不仅会回答人工智能所不能回答的问题,而且从长远来看,人类也会帮助改善人工智能技术,“M”除了能做到回答问题、查阅信息等基本功能,还可以帮助用户完成如购买商品、餐厅定位、安排旅行计划等操作。

2015年11月,谷歌开源了全新的TensorFlow机器学习系统,该系统更快、更智能、更具有弹性。2015年1月,机器学习平台GraphLab改名为Dato,并获得了1850万美元的新融资(投资方为Vulcan Capital、Opus Capital、New Enterprise Associates、Madrona Venture Group),此前他们曾获得680万美元的融资。

在2015年12月召开的“2015年神经信息处理系统”(NIPS)会议上,微软研究人员和工程师公开了20多篇机器学习最新研究成果的论文。此外,微软还宣布,机器学习正在成为Windows 10的一部分:Skype翻译可以将口语几乎实时地翻译成其他语言,就像《星际迷航》中的通用翻译器那样,可以做到面对面的交流。Cortana个人数字助理在与用户的互动中不断学习与改进,从而帮助用户管理日历、跟踪快递甚至与用户聊天和讲笑话,实现真正的个性化互动体验。Clutter是微软Office 2016的成员,通过学习它可以识别出哪些电子邮件对用户来说最重要,并自动将不重要的邮件重定向到一个单独的文件夹中,从而保持用户收件箱的整洁。

2015年年底,隶属于中国科学院的新松机器人自动化公司生产了智能复合型机器人,这个安装了眼睛和感知器件的智能机器人,可以在车间里自由地行走并十分精确地完成任务,当其他工位人手不足时,接到指令的他还会主动上前帮忙,马上进入角色并开始工作。

百度创造和完善了大规模机器学习的技术,搭建了一个能容纳万亿特征数据的、分钟级别模型更新的、高效训练的点击率预估系统;为进一步深入地发展机器学习技术,百度开始研究如何从“机器学习”到“复制人类大脑”;2016年年初,百度提出,百度的产品和服务都靠机器学习等技术来驱动。

### 1.1.3 机器学习的未来

纵观人类文明的发展史,人类已经走过了石器时代、红铜时代、青铜时代、铁器时代、黑暗时代、启蒙时代、蒸汽时代、电气时代、原子时代等时代历程,未来近百年内,人类将从原子时代走向智能时代,而且对于机器学习的未来,人类已经提出了很多构想。

2013年4月的汉诺威工业博览会上,工业4.0战略的概念被首次提出,“工业4.0”是指以智能制造为主导的第四次工业革命,或者革命性的生产方法。该战略旨在通过充分利用信息通信技术和网络空间虚拟系统与信息物理系统相结合的手段,将制造业向智能化转型。根据机器学习等智能技术的未来发展趋势,工业4.0主要包括智能工厂(重点研究智能化生产系统及过程,以及网络化分布式生产设施的实现)、智能生产(主要涉及整个企业的生产物流

管理、人机互动,以及3D技术在工业生产过程中的应用等)、智能物流(主要通过互联网、物联网、物流网,整合物流资源,充分发挥现有物流的效率)等应用。

2015年9月,美军军队医疗中心指挥官少将 Steve Jones 在美军陆军的一次会议上发言表示,未来可以让智能机器人代替人类上战场运送伤员,美国军方甚至高调宣布:未来战场上机器人救起的可能不是人,而是机器人,因为智能机器人军团将代替人类出征。

在21世纪以前,“人工智能大爆炸”的设想似乎还只是科幻小说家杞人忧天的幻想。到了今天,有越来越多的人开始严肃地思考一个问题:当技术奇点到来的时候,人类将会怎样?2009年,Kurzweil 与 X-Prize 创始人 Peter Diamandis 共同建立了奇点大学(Singularity University),致力于“聚集、教育并激励一批核心的领导者,以应对人类在指数级增长的科技下遭遇到的重要挑战”,人类保卫战似乎已经迫在眉睫。这所大学由谷歌、欧特克、美国基因技术公司等联合支持创建,共有三个项目,覆盖范围包括机器人学、医学、生物科技、数据科学和企业管理等。

云计算带来了强大的运算能力,大数据算法也得到了广泛应用,虽然它们还不足以使计算机变得更智能,但它们创造了强人工智能产生的必要条件——大数据使得机器能够从海量的信息中进行学习,云计算拥有廉价且强大的接近人脑的运算能力。

在不久的将来,机器学习将走向强人工智能时代,将出现真正的能推理和解决问题的智能机器,这些机器将有知觉和自我意识,智能水平与人类相当。

## 1.2 机器学习应用前景

机器学习应用广泛,无论是在军事领域还是民用领域,都有机器学习算法施展的机会。

### 1.2.1 数据分析与挖掘

“数据挖掘”和“数据分析”通常被相提并论,并在许多场合被认为是可以相互替代的术语。关于数据挖掘,现在已有多种文字不同但含义接近的定义,例如“识别出巨量数据中有效的、新颖的、潜在有用的、最终可理解的模式的非平凡过程”;百度百科将数据分析定义为:“数据分析是指用适当的统计方法对收集来的大量第一手资料和第二手资料进行分析,以求最大化地开发数据资料的功能,发挥数据的作用,它是为了提取有用信息和形成结论而对数据加以详细研究和概括总结的过程。”无论是数据分析还是数据挖掘,都是帮助人们收集、分析数据,使之成为信息,并作出判断,因此可以将这两项合称为“数据分析与挖掘”。

数据分析与挖掘技术是机器学习算法和数据存取技术的结合,利用机器学习提供的统计分析、知识发现等手段分析海量数据,同时利用数据存取机制实现数据的高效读写。机器学习在数据分析与挖掘领域中拥有无可取代的地位,2012年Hadoop进军机器学习领域就是一个很好的例子。

2012年,Cloudera收购Myrrix共创Big Learning,从此,机器学习俱乐部多了一名新会员。Hadoop和便宜的硬件使得大数据分析更加容易,随着硬盘和CPU越来越便宜,以及开



源数据库和计算框架的成熟，创业公司甚至个人都可以进行 TB 级以上的复杂计算。Myrrix 从 Apache Mahout 项目演变而来，是一个基于机器学习的实时可扩展的集群和推荐系统。

其他大公司也纷纷采用机器学习技术分析数据，以提高其产品和服务的质量。微软官方正式启动 Azure 机器学习平台，并已经在 Xbox 和 Bing 中使用，它能够支持 R、Python、Hadoop、Spark 等语言。福特用 AI 安排工作进度，通过 AI 解决其因员工日益增多而带来的工作安排问题。谷歌将大型机器学习技术应用于药物发现，将神经网络的深度学习应用于虚拟药物筛选，主要是试图替换或提高高通量筛选过程中的计算方法。雅虎使用机器学习算法挖掘 160 亿邮件数据，实验室的研究人员研究了两百万人之间的 160 亿封邮件，以分析用户的行为习惯。PayPal 使用机器学习来打击诈骗，通过机器学习和统计模型来识别诈骗行为，将更复杂的算法用于过滤交易。AWS 向欧洲开发商开放机器学习服务，可以通过 AWS Dublin 区域使用该服务，公司期望亚马逊的机器学习能够帮助解决限制问题，所有的分析和预测均通过在欧洲的数据完成，并且从不离开这个区域。

### 1.2.2 模式识别

模式识别起源于工程领域，而机器学习起源于计算机科学，这两个不同学科的结合带来了模式识别领域的调整和发展。模式识别研究主要集中在两个方面：一是研究生物体（包括人）是如何感知对象的，属于认识科学的范畴；二是在给定的任务下，如何用计算机实现模式识别的理论和方法，这些是机器学习的长项，也是机器学习研究的内容之一。

模式识别的应用领域广泛，包括计算机视觉、医学图像分析、光学文字识别、自然语言处理、语音识别、手写识别、生物特征识别、文件分类、搜索引擎等，而这些领域也正是机器学习大展身手的舞台，因此模式识别与机器学习的关系越来越密切，以至于国外很多书籍把模式识别与机器学习综合在一本书里讲述。

### 1.2.3 更广阔的领域

2015 年是机器学习年，用机器学习改造人类社会的革命时代已经来临。不但谷歌、亚马逊、埃森哲、丰田、特斯拉、美国强生等大公司都在大规模地采用机器学习技术，而且，创业公司也加入了这场机器学习的革命，并拥有和大公司同等的地位。创业公司已经公布了机器学习的创新型应用，投资商对机器学习创业公司表示出浓厚的兴趣。已经有超过 170 家创业公司进入 AI 浪潮，谷歌、IBM 等大型科技公司并对 AI 投入重金。

机器学习技术在全球各行各业的应用已经进入井喷时期，相关新闻比比皆是。Google DeepMind 研究人员已经成功让计算机通过机器学习成为 Atari 视频游戏的大师。谷歌与美国强生合作发展的 AI 手术机器人，能够帮助外科医生减少对病人的伤害。Facebook 开发了人工智能测试，能够判断 AI 的智能程度。Dato 也加入了机器学习创业洪流，机器学习平台 GraphLab（GraphLab 是一个开源项目，旨在帮助机器分析图像，如社交关系图）改名 Dato，获得了 1850 万美元的新融资。微软已经为“小娜”构建了聊天机制，机器学习使得“小娜”不仅能够识别玩笑并能预测运动赛事，还能够告诉你早点去开会（比如因为交通堵塞）。英

特尔的 18 核 Xeon 芯片为机器学习专门做了调整, 为快速变化的服务市场设计了 E7 芯片, 该公司宣称, 新的芯片在运作企业应用时要快 6 倍。Airbnb 公布了机器学习包 Aerosolve, Airbnb 相信人与机器以共生的方式进行合作的效率会比只有人或机器高, 这个包的设计本质是追求人性化。机器学习也进入了 Gartner 的 2015 Hype Cycle 报告, Hype Cycle 只展示数字人文主义的技术并且它主要展示 Gartner 认为有重大影响的技术, 而机器学习是报告中第一个出现的技术。

### 1.3 小结

机器学习作为一门多领域交叉学科, 该领域的主要研究对象是人工智能, 专门研究计算机怎样模拟或实现人类的学习行为, 以获取新的知识或技能, 重新组织已有的知识结构, 使之不断改善自身的性能, 它是人工智能的核心, 是使计算机具有智能的根本途径。

近年来, 机器学习的研究与应用在国内外越来越重视。机器学习已经广泛应用于语音识别、图像识别、数据挖掘等领域。大数据时代的到来, 使机器学习有了新的应用领域, 从包含设备维护、借贷申请、金融交易、医疗记录、广告点击、用户消费、客户网络行为等数据中发现有价值的信息已经成为其研究与应用的热点。

我们以记者与雨果·德·加里斯教授的部分专访内容来结束这一章。

记者: 为什么选择这个研究工作?

雨果·德·加里斯: 对人类大脑的好奇心和人脑的想象力的好奇心。人类只是一个个分子构成的机器, 像计算机的芯片一样, 像编制程序一样。另一方面, 作为生物人都会死亡消失的, 但人工智能机器就不会。所以说, 这个研究就像制造神一样。

当人类促使技术进步, 让具有人工智能的机器人得以诞生和发展, 但总有一天人工智能机器会实现自己进化, 当这种技术达到一个奇点的时候, 就不需要人类来推动了。比人类聪明得多的人工智能机器将在以年单位的短时间里产生。

记者: 预测一下未来人工智能机器的前景。

雨果·德·加里斯: 下一个 20 年, 它们很有可能出现在我们的家里, 为我们打扫房间, 照顾小孩, 和我们聊天, 给我们来自地球上知识库里面的无限知识。我们还将可以和它们有性关系, 被它们教育, 从它们那里得到娱乐和开怀大笑。20 年后的大脑制造业, 每年全球范围内将可能创造万亿美元的价值。人工智能机器有比我们聪明万亿倍的可能性, 不夸张地说, 人工智能机器和人类交流, 就像人类试图和岩石交流一样艰难。不过, 真正的人工智能在我死后的三四十年内不会被制造出来。我活着看不到工作的真正结果, 这是让我沮丧和失望的一个根源。

## 科学计算平台

机器学习算法具有坚实的数学理论支持，机器学习的应用建立在科学计算的基础上，而数学计算又是科学计算的主要组成部分。计算机技术的飞速发展和计算数学方法及理论的日益成熟，使解决复杂的数学计算问题成为可能。这些问题在以前用一般的计算工具来解决非常困难，而现在用计算机来处理却非常容易。目前用计算机处理得较多的数学计算主要分为以下两类：

第一类是数值计算，它以数值数组作为运算对象，给出数值解；计算过程中可能会产生误差累积问题，影响了计算结果的精确性；计算速度快，占用资源少。

第二类是符号计算，它以符号对象和符号表达式作为运算对象，给出解析解；运算不受计算误差累积问题的影响；计算指令简单；占用资源多，计算耗时长。

数值计算方法成为了科学计算的重要手段，它研究怎样利用计算工具来求出数学问题的数值解。数值计算方法的计算对象是微积分、线性代数、插值与逼近及最小二乘拟合、数值积分与数值微分、矩阵的特征值与特征向量求解、线性方程组与非线性方程求根，以及微分方程数值解法等数学问题，这些是模式识别、数据分析及自动制造等机器学习领域需要应用的数学。

符号计算是专家系统等机器学习领域需要应用的数学，在符号计算中，计算机处理的数据和得到的结果都是符号。符号既可以是字母和公式，也可以是数值，其运算以推理解析的方式进行，不受计算误差积累问题困扰，计算结果为完全正确的封闭解或任意精度的数值解，这意味着符号计算给出的结果能避免因舍入误差而引起的问题。还有更多的数学分支正在进入机器学习领域，复杂的数学计算需要强大的科学计算平台。科学计算平台提供了机器学习算法应用的底层支持。

## 2.1 科学计算软件平台概述

现代科学研究的方法主要有三种：理论论证、科学实验、科学计算。近年来，科学计算方法逐步成为科学研究的主流方法，在金融工程、信息检索、基因研究、环境模拟、数值计算、数据分析、决策支持等领域得到了广泛使用。由于计算机技术的发展及其在各技术科学领域的应用推广与深化，这些应用领域不论其背景与含义如何，都要用计算机进行科学计算，都必须建立相应的数学模型，并研究其适合于计算机编程的计算方法。科学计算平台已经成为科学研究必要的基础条件平台，有力地推动了科学研究的发展和工程技术的进步。

机器学习应用需要科学计算的支持。大部分科学计算应用的领域都需要用到机器学习算法，科学计算平台与机器学习之间的关系就像鱼与水的关系。现代机器学习研究与应用早已经离不开科学计算平台的支撑，科学计算平台也因为机器学习的迅猛发展而进入了全新的百家争鸣时代。

### 2.1.1 常用的科学计算软件

目前常用的科学计算软件有以下几种：

#### 1. MATLAB

MATLAB 是一种用于数值计算、可视化及编程的高级语言和交互式环境。使用 MATLAB，可以分析数据、开发算法、创建模型和应用程序，通过矩阵运算、绘制函数和数据、实现算法、创建用户界面、连接其他编程语言等方式完成计算，比电子表格或传统编程语言（如 C/C++ 或 Java）更方便快捷。MATLAB 具有强大的数值计算功能，可完成矩阵分析、线性代数、多元函数分析、数值微积分、方程求解、边值问题求解、数理统计等常见的数值计算，同时它也能进行符号计算。

#### 2. GNU Octave

GNU Octave 与 MATLAB 相似，它是自由软件基金会开发的一个自由再发布软件，以 John W. Eaton 为首的一些志愿者共同开发了叫作 GNU Octave 的高级语言，这种语言与 MATLAB 兼容，主要用于数值计算，同时它还提供了一个方便的命令行方式，可以数值求解线性和非线性问题，以及做一些数值模拟。

#### 3. Mathematica

Mathematica 系统是美国 Wolfram 研究公司开发的一个功能强大的计算机数学系统。它提供了范围广泛的数学计算功能，支持在各个领域工作的人们做科学研究的过程中的各种计算。这个系统是一个集成化的计算机软件系统，它的主要功能包括符号演算、数值计算和图形三个方面，可以帮助人们解决各种领域里比较复杂的符号计算和数值计算的理论和实际问题。

#### 4. Maple

1980 年 9 月，加拿大滑铁卢大学的符号计算研究小组研制出一种计算机代数系统，取名为 Maple，如今 Maple 已演变成成为优秀的数学软件，它具有良好的使用环境、强有力的符号



计算能力、高精度的数字计算、灵活的图形显示和高效的可编程功能。Maple 在符号计算方面功能强大，符号计算式可以直接以数学的形式来输入和输出，直观方便。

## 5. SPSS

SPSS 预测分析是 IBM 公司的产品，它提供了统计分析、数据和文本挖掘、预测模型和决策优化等功能。IBM 宣称，使用 SPSS 可获得 5 大优势：商业智能，利用强大而简单的分析功能，控制数据爆炸，满足组织灵活部署商业智能的需求，提升用户期望值；绩效管理，指导管理战略，使其朝着最能盈利的方向发展，并提供及时准确的数据、场景建模、浅显易懂的报告等；预测分析，通过发现细微的模式关联，开发和部署预测模型，以优化决策制定；分析决策管理，一线业务员工可利用该系统，与每位客户进行互动，从中获取丰富信息，提高业务成绩；风险管理，在合理的前提下，利用智能的风险管理程序和技术，制定规避风险的决策。

## 6. R

R 语言是主要用于统计分析、绘图的语言和操作环境。R 目前由“R 开发核心团队”负责开发，它是基于 S 语言的一个 GNU 项目，语法来自 Scheme，所以也可以当作 S 语言的一种实现，虽然 R 主要用于统计分析或者开发统计相关的软件，但也可用作矩阵计算，其分析速度堪比 GNU Octave 甚至 MATLAB。R 主要是以命令行操作，网上也有几种图形用户界面可供下载。R 内建多种统计学及数字分析功能，还能透过安装套件 (Packages) 增强。

## 7. NumPy、SciPy、matplotlib 等 Python 科学计算平台

Python 是一种面向对象的、动态的程序设计语言，它具有非常简洁而清晰的语法，既可以用于快速开发程序脚本，也可以用于开发大规模的软件，特别适合于完成各种高层任务。随着 NumPy、SciPy、matplotlib 等众多程序库的开发，Python 越来越适合用于科学计算。NumPy 是一个基础科学的计算包，包括：一个强大的 N 维数组对象封装了 C++ 和 Fortran 代码的工具、线性代数、傅立叶转换和随机数生成函数等其他复杂功能的计算包。SciPy 是一个开源的数学、科学和工程计算包，能完成最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解等计算。matplotlib 是 Python 最著名的绘图库，它提供了一整套和 MATLAB 相似的命令 API，十分适合交互式制图，它也可以方便地用作绘图控件，嵌入 GUI 应用程序中。

### 2.1.2 本书使用的工程计算平台

MATLAB、Mathematica、Maple、SPSS 等软件功能齐全、界面友好，同时内含多种强大的软件包，但价格昂贵，它们都是商业化软件，GNU Octave、R 和 Python 科学计算包作为开源免费的工程计算平台，会是不错的选择。

本书选择 R 和 Python 科学计算包作为工程计算平台，Python 和 R 都能在多种操作系统平台上运行。Python 是一种非常流行的脚本语言，用户较多，容易掌握，也有成熟并行计算框架 dispy 等，测试成功的单机机器学习算法稍加修改就能应用于大规模分布式计算的工程之中。正是因为 Python 具有如此之多的优点，Google 内部也经常使用它。R 语言内置大量统

计分析包，能访问部分系统函数，核心为解释执行的语言，大部分用户可见的 R 函数由 R 语言本身编写，出于效率原理，计算密集型任务通过在运行时链接与调用 C、C++、FORTRAN 代码完成。此外，通过 Rcpp 能把丰富的 R 环境与 C/C++ 等结合，将 R 的 API 与数据对象封装成类以及类的方法，供外部 C++ 程序调用。

## 2.2 计算平台的配置

本章将以 Windows 平台和 Linux 平台为例，讲解 R 和 Python 科学计算平台的配置。Python 和 R 具有跨平台运行的特点，Windows 平台编写的 Python 和 R 代码只需修正兼容性问题即可正常运行在类 UNIX 平台上，如：中文字符的 UTF8 与 GBK 转换、Windows 系统与类 UNIX 平台的文件路径差异等。

### 2.2.1 Numpy 等 Python 科学计算包的安装与配置

Python 科学计算包有两种安装方式，即：分别安装科学计算平台内的软件包和安装 WinPython 集成计算包。如果是开发环境，建议使用 WinPython。

#### 1. 分别安装科学计算平台内的软件包

先安装 Python，关于它的版本，推荐使用 2.7 版本，然后安装 NumPy、SciPy、matplotlib 等 Python 软件包，它们都有 Windows 系统下的安装包。

Python 安装包的下载页面为 <http://www.python.org/download/>，选择 2.7 版本的 Windows 安装可执行文件下载即可。

NumPy 安装包下载页面为 <https://pypi.python.org/pypi/numpy>，下载 Windows 版本的安装可执行文件即可。

SciPy 安装包下载页面为 <https://pypi.python.org/pypi/scipy/>，该软件包目前没有 Windows 版本的安装执行文件，要用传统的 Python 安装第三方软件包的方式安装，将安装包下载解压，然后在命令行进入解压目录，输入以下命令：

```
python setup.py install
```

Matplotlib 软件包的下载页面为 <http://matplotlib.org/downloads.html>，下载 Windows 版本的安装可执行文件即可，注意应下载 Latest stable version 对应的软件包。Windows 版本的安装可执行文件通常命名格式为：产品名称 + 平台名称 + CPU 型号 + 版本号。以 Matplotlib 为例，打开其下载页面，如图 2-1 所示。

假设计算机的操作系统是 32 位，Python 版本号为 2.7，则下载安装 matplotlib-1.3.0.win32-py2.7.exe，如果操作系统是 64 位的，Python 版本号为 2.7，则下

#### Downloads

##### 1.3.0 — Latest stable version

- [matplotlib-1.3.0.tar.gz](#)
- [matplotlib-1.3.0.win-amd64-py2.6.exe](#)
- [matplotlib-1.3.0.win-amd64-py2.7.exe](#)
- [matplotlib-1.3.0.win-amd64-py3.2.exe](#)
- [matplotlib-1.3.0.win-amd64-py3.3.exe](#)
- [matplotlib-1.3.0.win32-py2.6.exe](#)
- [matplotlib-1.3.0.win32-py2.7.exe](#)
- [matplotlib-1.3.0.win32-py3.2.exe](#)
- [matplotlib-1.3.0.win32-py3.3.exe](#)

图 2-1 Matplotlib 下载页面

载安装 matplotlib-1.3.0.win-amd64-py2.7.exe。

在类 UNIX 平台上 (以 UBUNTU 为例), 可使用下面的命令安装 Python 及相关科学计算包:

```
sudo apt-get install python-numpy python-scipy python-matplotlib ipython
ipython-notebook python-pandas python-sympy python-nose
```

## 2. 安装 WinPython 集成计算包

WinPython 集成计算包集成了 Numpy 等第三方 Python 科学计算库, 在 winpython.sourceforge.net 下载并安装 WinPython 后, Numpy 等计算库和 Python 2.7 会一同被安装。此外, WinPython 附带一款非常不错的 IDE 开发调试环境: Spyder, 如图 2-2 所示是 Spyder 的界面截图。

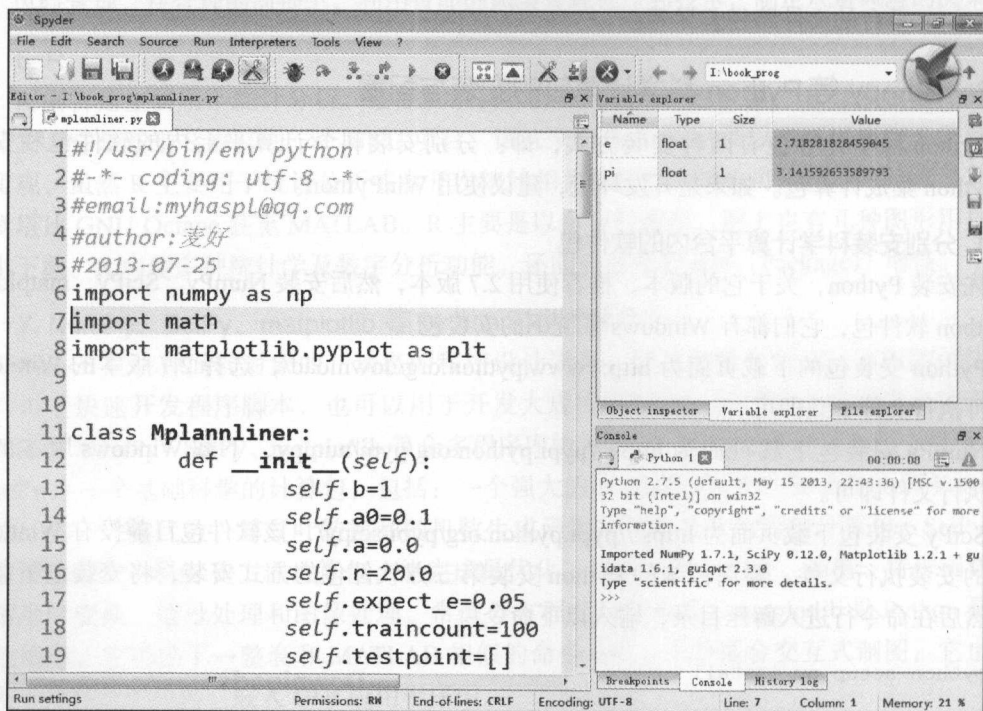


图 2-2 Spyder 界面

在图 2-2 所示的界面中, 右上角是类似于 MATLAB 的“工作空间”, 可很方便地观察和修改变量 (包含多维数组) 的值, 同时还拥有方便用户的智能代码 (Call-Tips 和 Auto-Complete) 功能, 如图 2-3 所示。

在 IDE 开发窗口下方的 Console 栏可以使用 pdb (类似于 C 语言的 GDB 调试工具) 调试 Python 代码, 也可以通过 Spyder 的调试菜单进行调试。下面是 pdb 调试工具的使用帮助:

```
>>> debugfile(r'K:\book_prog\zxecf.py', wdir=r'K:\book_prog')
> k:\book_prog\zxecf.py(7)<module>()
```

```
-> import matplotlib.pyplot as plt
(pdb) help
Documented commands (type help <topic>):
```

```
=====
EOF      bt          cont      enable   jump     pp        run       unt
a        c          continue exit     l        q         s         until
alias   cl         d        h        list    quit     step      up
args    clear      debug    help     n        r         tbreak    w
b        commands  disable ignore    next    restart  u         whatis
break   condition  down    j         p        return   unalias   where
```

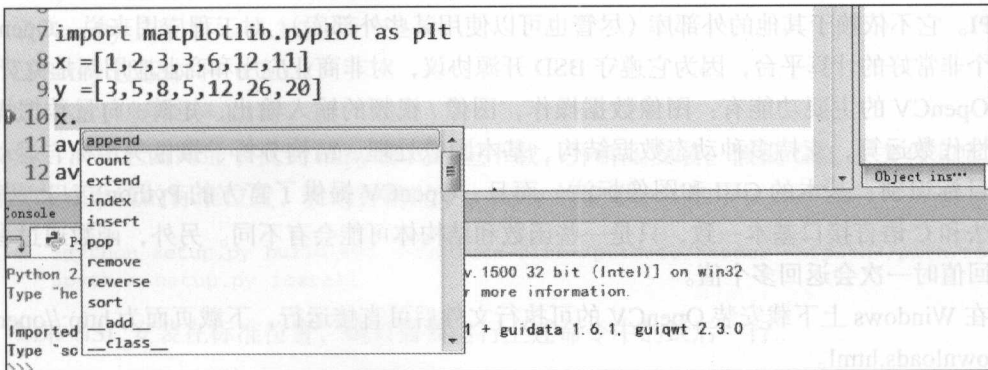


图 2-3 智能代码功能

常用的 pdb 调试命令如下：

- ❑ h(elp): 打印当前版本 pdb 可用的命令。
- ❑ disable/enable: 禁用 / 启用断点。
- ❑ n(ext): 让程序运行下一行。
- ❑ c(ontinue): 让程序正常运行，直到遇到断点。
- ❑ j(ump): 让程序跳转到指定的行数。
- ❑ b(reak): 设置断点，例如“b 23”，就是在当前脚本的 23 行打上断点，函数名也可作为参数。
- ❑ condition: 设置条件断点。下面语句就是对第 5 个断点加上条件  $x \geq 8$ :

```
(Pdb) condition 5 x >= 8
```

- ❑ cl(ear): 清除指定参数的断点或所有断点。
- ❑ p: 打印某个变量。比如：

```
(Pdb) p _file
u' ./pic/dog.jpg'
```

- ❑ !: 感叹号后面跟着语句，可以直接改变某个变量。
- ❑ q(uit): 退出调试。



综上所述, 在 Spyder 的帮助下, 能更高效地开发与调试 Python 代码, 因此笔者推荐在开发环境中安装 WinPython, 方便快捷, 有利于机器学习算法代码的编写。此外, 安装好 WinPython 后, 请在 windows 操作系统的“开始→程序→WinPython→WinPython Control Panel”中进行注册。

## 2.2.2 OpenCV 安装与配置

OpenCV 是 Intel 开源计算机视觉库, 它由一系列 C 函数和少量 C++ 类构成, 实现了图像处理和计算机视觉方面的很多通用算法。OpenCV 拥有包括 300 多个 C 函数的跨平台的中高层 API。它不依赖于其他的外部库 (尽管也可以使用某些外部库), 对工程应用来说, OpenCV 是一个非常好的计算平台, 因为它遵守 BSD 开源协议, 对非商业应用和商业应用都是免费。

OpenCV 的主要功能有: 图像数据操作, 图像/视频的输入输出, 矩阵/向量数据操作及线性代数运算, 支持多种动态数据结构、基本图像处理、结构分析、摄像头定标、运动分析、目标识别、基本的 GUI 和图像标注。而且, OpenCV 提供了官方的 Python 接口, 其使用方法和 C 语言接口基本一致, 只是一些函数和结构体可能会有不同。另外, 函数通过参数来返回值时一次会返回多个值。

在 Windows 上下载安装 OpenCV 的可执行文件后可直接运行, 下载页面为 <http://opencv.org/downloads.html>。

其在 Linux 平台上的安装方式在 OpenCV 官网上有介绍, 具体安装顺序如下:

- 1) 安装基本软件包。GCC 4.4.x 或更高版本、CMake 或更高版本、Git、GTK+2.x 或更高版本、including headers (libgtk2.0-dev)、pkgconfig、Python 2.6 或更高版本、Numpy 1.5 或更高版本、python-dev、python-numpy、ffmpeg 或 libav 开发包、libavcodec-dev、libavformat-dev、libswscale-dev。
- 2) 安装可选软件包。libdc1394 2.x、libjpeg-dev、libpng-dev、libtiff-dev、libjasper-dev。
- 3) 在 <http://opencv.org/downloads.html> 下载其源代码, 解压后, 进入目录以源代码编译方式安装 OpenCV。

```
$cd ~/opencv
$mkdir release
$cd release
$cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
$make
$sudo make install
```

OpenCV 官方提供了 Python 绑定库, 以 Python2.7 为例讲述了安装绑定库的方法, 在 Windows 下, 将它复制到 Python 的目录下, 将 opencv\build\python\2.7 下的 cv2.pyd 文件复制到 python-2.7.5\Lib\site-packages 目录下即可。在 Linux 下安装了 Python 后, 要确保 usr/lib/python2.7/site-packages 下有 cv.py 和 cv2.so 文件, 如果没有, 将这两个文件复制过来即可。

## 2.2.3 mipy 安装与配置

mipy 是基于 NumPy/SciPy 和 GSL 构建的 Python 模块, 它提供了高层函数和类, 允许

使用少量代码来完成复杂的分类、特征提取、回归、聚类等任务。mly 为免费软件，建立在 GPL3 开源协议之上。

mly 在 Windows 下的安装方式较简单，可以直接在下面网址下载可执行文件安装：

<http://sourceforge.net/projects/mlpy/files/>

在类 Linux 平台上，其安装方法稍稍复杂一些，以 Linux、OSX 和 FreeBSD 为例，安装配置 mly，需要先安装配置好以下软件：

□ GCC

□ Python 且版本  $\geq 2.6$  或为 3.X

□ NumPy 且版本  $\geq 1.3.0$

□ SciPy 且版本  $\geq 0.7.0$

□ GSL 且版本  $\geq 1.11$

然后，在上面网址中找到 mly 源代码包下载，并解压安装。假设 GSL 头文件和库文件没有安装在系统的标准位置，在这种情况下，mly 的安装方式如下：

```
$python setup.py build_ext --include-dirs=/path/to/header --rpath=/path/to/lib
$python setup.py install
```

如果 GSL 安装在标准位置，则只需要运行上述命令中的最后一行。

## 2.2.4 BeautifulSoup 安装与配置

BeautifulSoup 是用 Python 写的一个 HTML/XML 的解析器，它可以很好地处理不规范标记，并生成剖析树，通常用来分析“爬虫”抓取的 Web 文档，或者直接充当部分“爬虫”的角色。对于不规则的 HTML 文档，有补全功能。有了它，解析与分类网页就方便多了，节省了开发者的很多时间和精力。

安装 BeautifulSoup 很简单，在 Windows 平台和 Linux 平台上都是使用的传统第三方库安装方式。首先下载 BeautifulSoup 源码，其官网为：<http://www.crummy.com/software/BeautifulSoup/>。

然后解压后运行以下命令：

```
python setup.py install
```

此外，在 UBUNTU 下还可以使用系统包管理器安装。

```
$ apt-get install python-bs4
```

## 2.2.5 NeuroLab 安装与配置

NeuroLab 是一个简单而强大的、用 Python 编写的神经网络库，包括基础神经网络、训练算法，并具有弹性的构架，可创建其他网络，它用纯 Python 和 numpy 写成。API 的使用与 MATLAB 的神经网络工具箱类似，具有弹性的网络配置和学习算法，可以改变神经网络和学习算法的类型、训练、误差、初始函数和激活函数等神经网络参数。

Windows 和 Linux 下的安装方式如下。

首先下面的页面下载 Neurolab:

<http://code.google.com/p/neurolab/downloads/list>

然后解压后运行如下命令:

```
python setup.py install
```

## 2.2.6 R 安装与配置

R 的原始码可自由下载使用, 也有已编译的执行档版本可以下载, 可在多种平台下运行, 包括类 UNIX (包含 FreeBSD 和 Linux)、Windows 和 MacOS。

WINDOWS 安装方式如下。

首先访问其官网下载页面:

<http://ftp.ctex.org/mirrors/CRAN/>

然后下载安装可执行文件安装即可。

UBUNTU 下的安装方式如下:

```
$ sudo apt-get update
$ sudo apt-get install r-base
$ sudo apt-get install r-base-dev
```

## 2.3 小结

“不要重复造轮子”(Stop Trying to Reinvent the Wheel), 这可能是每个软件工程师入行时被告知的第一条准则。在轮子适合“机器学习”这台车的情况下, 机器学习算法才能跑得更好, 适合的科学计算平台就是机器学习的“轮子”。

笔者认为, 作为机器学习这驾马车的“轮子”应该具备以下特征:

- ❑ 开源免费, 且开源协议友好, 例如: LGPL 协议或 BSD 协议, 这样更有利于商业应用。
- ❑ 平台有文档, 接口规范, 能实际代码用例最好。
- ❑ 配置简单灵活, 支持的操作系统平台多, 运行速度快。
- ❑ 代码结构清晰、简单, 便于使用者修改这个“轮子”, 通俗地说: 移植性强。

本书采用的计算平台在本章都一一列出其安装和配置方法。算法是一种计算思维的描述, 万变不离其宗, 好的工具原理都差不多。

也许随着时间的推移, 算法在改进, 更好的“轮子”将出现, 所以不一定采用本书上所写的这些平台作为机器学习实验和应用的工具, 但有一条原则: 功能强大的计算平台不一定适合所有的工程, 一切以适用为准。

## 第二部分 Part 2 计算平台应用实例

## 基础篇

## 3.1 Python 计算平台简介及应用实例

目前, 科学计算平台很多, 在本节中, 使用 Python 编写的机器学习算法应用到计算平台; Numpy 等科学计算包, OpenCV 的 Python 绑定库, mipy 机器学习包, Beginning2oop 阿夏神经网络库, Neural 神经网络库等, 所有平台均为开源免费软件。本章将介绍这些计算平台的一些基础应用。

## 3.1.1 Python 语言基础

1990 年圣诞节期间, 吉多·范罗苏姆为了打发假日时间, 设计了一个新的脚本解释器, 作为 ABC 语言的一种继承, 就这样, Python 诞生于荷兰。Python 的设计思想是简单、明确、简单。Python 提供了丰富的 API 和工具, 使程序员能使用 C 语言、C++、Python 解释器本身也可以被集成到其他需要脚本语言的程序中。此外, Python 语言具有可读性、易学性以及可移植性, 因此国内外用 Python 做科学计算的科研机构、商业公司日益增多, 比如三大经典科学计算库 NumPy、SciPy 和 Matplotlib 就是 Python 语言实现的。Python 语言具有可读性、易学性以及可移植性, 因此国内外用 Python 做科学计算的科研机构、商业公司日益增多, 比如三大经典科学计算库 NumPy、SciPy 和 Matplotlib 就是 Python 语言实现的。

——老子

合抱之木, 生于毫末; 九层之台, 起于累土; 千里之行, 始于足下。

## Chapter 3 第 3 章

# 计算平台应用实例

## 3.1 Python 计算平台简介及应用实例

目前,科学计算平台很多,在本书中,使用 Python 编写的机器学习算法用到的计算平台有: Numpy 等科学计算包、OpenCV 的 Python 绑定库、mlpy 机器学习库、BeautifulSoup 网页解析库、Neurolab 神经网络库等,所有平台均为开源免费软件。本章将讲解这些计算平台的操作,并解析一些基础实例应用。

### 3.1.1 Python 语言基础

1989 年圣诞节期间,吉多·范罗苏姆为了打发假日时间,决心开发一个新的脚本解释程序,作为 ABC 语言的一种继承,就这样,Python 在吉多手中诞生了。Python 的设计哲学是优雅、明确、简单。Python 提供了丰富的 API 和工具,使程序员能使用 C 语言、C++、Cython 编写扩充模块。Python 编译器本身也可以被集成到其他需要脚本语言的程序中。此外,很多人把 Python 作为一种“胶水语言”使用,用它将其他语言编写的程序进行集成和封装。

Python 包含了一组完善而且容易理解的标准库,能够轻松地完成很多常见的任务,且代码语法简洁、清晰,使用缩进定义语句块,具备很高的可读性。由于 Python 语言具有简洁、易读以及可扩展性,在国内外用 Python 做科学计算的研究机构、商业公司日益增多,比如:三大经典科学计算库 Numpy、SciPy 和 matplotlib 均扩展了 Python,通过 Python 可以轻松完成快速数组处理、数值运算和绘图任务。

#### 1. Python 基本数据类型

Python 是解释运行的动态语言,解释器的提示符为“>>>”。Python 的基本数据类型包括数字型、字符串型和列表,此外还可以用类型表示函数、模块、类型本身、对象的方法、



编译后的 Python 代码、运行时信息等。

1) 数字型, 可将 Python 作为一个计算器使用, 在计算过程中可使用 “+” (加)、“-” (减)、“\*” (乘)、“/” (除)、“(”、“)” 以及 “%” (取余) 等操作符。此外, Python 用 “#” 表示其后的内容是注释。下面代码演示了其基本的计算功能和注释的使用。

```
>>> 2+2
4
>>> # 本行是注释
... 2+8
10
>>> (50-5*6)/2
10
>>> 17/-4
-5
>>> 9%2
1
```

Python 使用 “=” 进行赋值操作, 赋值操作不会返回任何结果, 也可以在同一行中连接赋值, 赋值语句将从右到左依次完成赋值。下面的代码对变量进行复数和实数的赋值。

```
>>> c=5.2-6.5j### 复数
>>> c
(5.2-6.5j)
>>> c.real### 实部
5.2
>>> c.imag### 虚部
-6.5
>>>x = y = z = 0
```

变量在使用前, 要处于定义状态 (即已经赋值), 否则会出错。下面代码试图对未定义的变量 myx 进行操作结果出错了。

```
>>> myx # 访问未定义的变量
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'myx' is not defined
```

在 Python 计算中可使用浮点数。如果在计算过程中出现了浮点数, 则整型会自动转换为浮点型计算; 如果全是整型, 则计算结果也为整型。在下面例子中, 第一行代码两个操作数均为整型, 返回的结果并不会精确为整型; 而在第二行代码中, 第一个操作数 7.0 为浮点型, 返回的结果为浮点数。(以下代码请在 Python 解释器下运行)。

```
>>> 7/3
2
>>> 7.0/3
2.3333333333333335
```

复数运算使用 (real+imagj) 的形式, 也可使用 complex(real,imag) 创建一个复数对象, 其中 real 表示实部, imag 表示虚部。下面是一些复数计算的例子。

```

>>> 2+6j
(2+6j)
>>> 2-6j
(2-6j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> a=1.5+0.5j
>>> a.real
1.5
>>> a.imag
0.5

```

变量“\_”表示刚计算的结果。下面代码通过计算  $(3+2) \times 6$  演示了该变量的使用。

```

>>> 3+2
5
>>> _*6
30

```

2) 字符串型。Python 的字符串通常用单引号和双引号包围的形式表示，通过 print 语句可将字符串输出到输出设备中（默认情况下输出到屏幕）。此外，Python 2.0 及以后的版本支持 Unicode 字符串，中文字符一般使用 Unicode 编码（国际组织制定的容纳世界所有文字和符号的字符编码方案，用 0 ~ 0x10FFFF 映射字符，最多可以容纳 1 114 112 个字符），在前面加 u 表示后面是 Unicode 字符串。下面的代码演示了 Unicode 字符的定义与使用、字符串的输出等操作。

```

>>> 'spam eggs'
'spam eggs'
>>> "Yes," he said.
'Yes," he said.'
>>> print u"你好"
你好
>>> print u"机器学习"
机器学习

```

Python 的字符串可视为列表（数组），使用“[索引]”的方式能对它进行切片操作（索引从 0 开始），使用“+”可对字符串进行连接。下面的代码演示了字符串的连接、切片等操作。

```

>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
'<HelpAHelpAHelpAHelpAHelpA>'
>>> word[-2:]      # 最后 2 个字符
'pA'
>>> word[:-2]      # 除去最后 2 个字符以外的字符
'Hel'

```

Python 的字符串可使用转义字符，方法是在特殊字符前加上“\”。主要的转义字符有：

- \' 单引号
- \" 双引号
- \a 发出系统响铃声
- \b 退格符
- \n 换行符
- \t 横向制表符
- \v 纵向制表符
- \r 回车符
- \f 换页符
- \\ \
- \o 八进制数代表的字符
- \x 十六进制数代表的字符
- \000 终止符，\000 后的字符串全部忽略

此外，Python 的字符串相比其他程序语言多了一种描述方式，就是用 3 个引号标示字符串，其功能是将字符串内容原样输出，如果字符串本身包括换行，则输出换行，如果包括特殊字符，则字符串无需使用转义字符。下面的代码演示了中文字符串的输出、转义字符的使用、字符串切片、统计长度、三引号使用等操作。

```
>>> 'doesn\'t'
"doesn't"
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> print u"你好,Python\n机器学习"
你好,Python
机器学习
>>> print u"""你好,Python
... 机器学习"""
你好,Python
机器学习
>>> mystr= u"你好,Python\n机器学习"
>>> print mystr
你好,Python
机器学习
>>> print mystr[:5]
你好,Py
>>> print mystr[3:5]
Py
>>> len(mystr)###len 函数计算字符串长度
14
```

Python 字符串的三引号表示方式意义重大，用它可以在 CGI（CGI 允许 Web 服务器执行外部程序，并将它们的输出发送给 Web 浏览器）程序中轻松输出 HTML 代码。下面是 CGI 的“Hello World”程序：

```
#!/usr/bin/env python
print "Content-Type: text/html"
print
print """<html>
<body>
<h2>Hello World!</h2>
</body>
</html>
"""
```

3) 列表。Python 可将不同类型(包括复合类型本身)的值组成复合类型,列表就是复合类型之一。与字符串相同,列表可以使用切片操作,其索引也是从 0 开始的。此外,统计列表的长度使用 len 函数,使用 “\*” 将生成新列表,其元素由源列表重复填充,使用 “+” 可连接列表。下面的代码演示了列表的定义、连接、重复填充、切片、统计长度等操作。

```
>>> a = ['hello', 'world', 100, 1234]
>>> a
['hello', 'world', 100, 1234]
>>> a[:2] + ['- ', 2*2]### 连接列表
['hello', 'world', '- ', 4]
>>> mylist=[1,23,45]
>>> mylist
[1, 23, 45]
>>> mylist*2### 重复填充列表
[1, 23, 45, 1, 23, 45]
>>> mylist[:2]### 列表切片
[1, 23]
>>> x = [12, 13]
>>> y = [11, x, 14]
>>> len(y)### 求列表长度
3
>>> y[1]
[12, 13]
>>> y[1][0]
12
```

列表和字符串也可称为序列,序列由若干个元素组成,每个元素的先后顺序明确,不能更改。

## 2. Python 语句

1) 条件语句。if 语句的作用是判断条件是否成立,如果成立,则执行后面的语句块; if ... elif ... elif 语句可用于对多个条件进行判断,并执行最先满足条件的语句块; else 语句表示所有条件都不成立时执行。下面的例子展示了 if 语句的使用方法,功能是对输入数字的范围进行判断,并将其中的负数转变为 0。

```
>>> x = int(raw_input("Please enter an integer: "))
Please enter an integer: -10
>>> if x < 0:
...     x = 0
```



```

... print 'Negative changed to zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Single'
... else:
...     print 'More'
Negative changed to zero
下面例子演示了判断非奇偶数。
... for x in range(1, 10):
...     if x % 2 == 0:
...         print x, 'is an even number'.
...     else:
...         # loop fell through without finding a factor
...         print x, 'is an odd number.'
2 is an even number
3 is an odd number
4 is an even number
5 is an odd number
6 is an even number
7 is an odd number
8 is an even number
9 is an odd number

```

2) 循环语句。for 语句表示循环，它与 C 语言中的 for 语句略有不同。C 语言的 for 语句可定义步长和终止循环条件，而 Python 的 for 语句在 Python 的序列（列表、字符串等）中迭代，每次只操作其中一项。下面的代码在单词列表中迭代，每次迭代输出单词及其长度。

```

>>> # Measure some strings:
... words = ['cat', 'window', 'hello']
>>> for w in words:
...     print w, len(w)
...
cat 3
window 6
hello 5

```

也可以在迭代过程中修改序列。下面的例子运行结果是在列表迭代过程中修改了列表本身。

```

>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words[:]:
...     if len(w) > 6:
...         words.insert(0, w)
...
>>> words
['defenestrate', 'cat', 'window', 'defenestrate']

```

3) range 函数。for 语句仅能在列表等序列中进行迭代，从表面上来看，它比 C 语言的 for 循环功能弱很多，其实不然，有了 range 函数，其功能远比 C 语言的 for 循环强大。range

函数可产生符合某种规律的列表等序列。下面代码产生 0 ~ 9 共 10 个数字, 增长步长为 1。

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

**range** 函数还可以产生更复杂的序列, 常用调用格式为:

```
range(起始值, 终止值, 步长)
```

其中起始值和步长可以省略, 起始值默认为 0, 步长默认为 1。

```
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, -100, -30)
[-10, -40, -70]
```

下面的代码是 **range** 函数与 **for** 语句组合的应用, 输出列表元素的索引及值。

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

下面的代码完成 1 至 10 中奇数的累加。

```
>>> mysum=0
>>> for i in range(1,10,2):
...     mysum=mysum+i
...
>>> mysum
25
```

4) **break** 与 **continue**。**break** 语句结束本层循环, **continue** 语句忽略下面的语句继续本层的下次循环。下面的代码使用 **break** 语句, 查找 2 至 10 以内 (不含 10) 的素数, 如果不是素数, 则分解因数。

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...         if n==x+1:
...             # loop fell through without finding a factor
...             print n, 'is a prime number'
2 is a prime number
3 is a prime number
```

```

4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3

```

下面的代码使用了 `continue` 语句, 判断 50 至 60 之间的数哪些是奇数, 哪些是偶数。

```

>>> for num in range(50, 60):
...     if num % 2 == 0:
...         print "偶数", num
...         continue
...     print "奇数", num
...
偶数 50
奇数 51
偶数 52
奇数 53
偶数 54
奇数 55
偶数 56
奇数 57
偶数 58
奇数 59

```

5) `while` 循环。在 `while` 循环中会一直执行后面的语句块, 直到条件不满足才终止。下面的代码用于在列表中循环, 并输出元素。

```

>>> a=range(10)
...
i=0
>>> while i<10:
...     print a[i],
...     i=i+1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

6) 函数定义。Python 使用 `def` 关键字定义函数。下面的代码定义了屏幕输出函数 `show`, 参数是要输出的内容。

```

>>> def show(mess="hello"):
...     print mess
...
>>> show()
hello
>>> show("机器学习")
机器学习

```

下面的代码定义了斐波那契数列的计算函数 `fib`。

```

>>> def fib(n):
...     """Print a Fibonacci series up to n."""

```

```

...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597

```

### 3. Python 的元组、集合以及字典

1) tuple 元组。tuple 元组类似列表，不同的是它的内容不能修改。Python 元组的定义方式是使用圆括号包含元素或直接列举元素。下面的代码将定义 x 为元组类型，当对 x[0] (x 的第一个元素) 进行修改时，Python 解释器提示错误。

```

>>> x = 10, 20, 'learn'
>>> x[0]
10
>>> x
(10, 20, 'learn')
>>> x1, x2, x3 = x
>>> x1
10
>>> x2
20
>>> x3
'learn'
>>> x[0] = 90
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>

```

2) Sets 集合。Python 使用方括号定义 Sets 集合，集合的特点是无重复元素，集合中的元素无先后顺序，也不能用索引进行管理。下面的代码定义了一个水果列表，通过 set 函数转换为集合，去除重复元素。

```

>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])

```

下面的代码通过 in 操作检查成员与集合的关系。

```

>>> 'orange' in fruit ### 集合类是否有成员 'orange'
True
>>> 'crabgrass' in fruit ### 集合类是否有成员 'crabgrass'
False

```

既然是集合，当然能对它进行集合的数学运算。Python 集合支持 union (联合或并)、intersection (交)、difference (差) 和 symmetric difference (对称差) 等操作，可通过“-”、“|”、“&”、“^”操作符计算集合的差集、并集、交集和对称差集。下面的代码演示了对集合 a 和 b 的运算。



```

>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                    # 差集
set(['r', 'd', 'b'])
>>> a | b                    # 并集
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                    # 交集
set(['a', 'c'])
>>> a ^ b                    # 对称差集
set(['r', 'd', 'b', 'm', 'z', 'l'])

```

3) Dictionaries 字典。Dictionaries(字典)是一种散列结构,可理解为 Hash(散列)类型。字典由若干个键值对组成,键值对是一种映射,一个键对应于一个值。键值对由两个部分组成,第一部分是键,键在字典中必须保持唯一,字典与列表和元组不同,列表和元组属于序列,元素以先后顺序来管理,而字典的元素是以键为单位对值进行管理的;第二部分为值,值与键一一对应,值可以彼此相同。

Python 使用花括号定义字典,使用类似索引的方式存取值,但索引为键。此外,可使用 del 操作删除键值对,使用 keys() 方法返回字典变量存储的所有键。下面的代码演示了一个学生学号的字典变量,以及如何对学生信息进行删除、查询等操作。

```

>>> tel = {'张三': 4098, '李四': 4139}
>>> tel
{'\xd5\xc5\xc8\xfd': 4098, '\xc0\xee\xcb\xc4': 4139}
>>> tel['张三']
4098
>>> del tel['张三']
>>> tel['张三']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: '\xd5\xc5\xc8\xfd'
>>> tel.keys()
['\xc0\xee\xcb\xc4']
>>> tel['王华']=1000
>>> tel.keys()
['\xcd\xcf5\xbb\xaa', '\xc0\xee\xcb\xc4']
>>> print tel.keys()[0]
王华

```

上面代码中,“\xcd”等为汉字在 Python 内部的编码。

#### 4. Python 类

Python 语言有强大的面向对象编程能力。Python 和 C++ 等语言一样拥有类机制,Python 类的定义方式如下:

```

class 类名:
    # 类成员变量
    变量 A=A 初始值

```

变量 B=B 初始值

```
.....
# 下面定义了类成员函数
def __init__(self, 参数 1, 参数 2, ..., 参数 n):
    # 类构造函数
.....
def __del__(self):
    # 析构函数
.....
def 方法 1(self, 参数 1, 参数 2, ..., 参数 n):
    # 类的方法
.....
.....
```

下面的代码定义了一个复数类 `Complex`，同时定义了类的实例变量 `x`，最后输出该变量的实部和虚部。`Complex` 类很简单，在类构造函数中对实部成员和虚部成员进行赋值。

```
>>> class Complex:
...     r=0
...     i=0
...     def __init__(self, realpart, imagpart):
...         self.r = realpart
...         self.i = imagpart
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

## 5. Python 异常处理

Python 的异常处理能力很强大，可准确反馈出错信息。在 Python 中，异常是对象，可对它进行操作。所有异常都是基类 `Exception` 的成员，从基类 `Exception` 继承，在 `exceptions` 模块中定义。Python 异常处理的格式如下：

```
try :
    # 下面为可能发生异常的语句块
    .....
except 异常类型 :
    # 下面为处理异常的语句块
    .....
```

下面的代码将检查输入是否为有效数字。程序通过将输入转换成整型来测试是否为数字，如果不是数字，`int` 函数将触发异常 `ValueError`，异常处理程序提示“哦！输入不是有效数字，请重新输入（Oops! That was no valid number. Try again...）”，直到输入正确格式的数字后，程序才退出。

```
>>> while True:
...     try:
...         x = int(raw_input("Please enter a number: "))
...         break
...     except ValueError:
```

```
... print "Oops! That was no valid number. Try again..."
...
```

前面演示了异常的被动触发，Python 还能主动触发异常，处理方式为：先通过 raise 语句抛出异常，然后用 except 捕捉异常。下面的代码演示 raise 主动抛出 NameError 异常后被捕捉，输出 “An exception flew by!” 后，继续抛出异常，以便给更外层的异常处理函数继续处理。

```
>>> try:
...     raise NameError('HiThere')
... except NameError:
...     print 'An exception flew by!'
...     raise
...
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: HiThere
```

以上简要介绍了 Python 编程语言的基本语法，它和 C++ 有几分相似，有一定编程基础的读者应该都能看懂。Python 是一门上手很快的编程语言，它与其他语言最大的不同就是它区分语句块时使用的不是括号，而是每行语句前面的空格数量。因此，Python 代码非常工整和漂亮，它严格遵守代码语句块的缩进原则，可依靠缩进判断语句块的范围。

### 3.1.2 Numpy 库

本书中介绍的机器学习算法大部分是调用 Numpy 库来完成基础数值计算的。下面了解一下 Numpy 库的基本使用方法。

#### 1. ndarray 数组基础

Python 中用列表保存一组值，可将列表当成数组使用。此外，Python 有 array 模块，但它不支持多维数组，无论是列表还是 array 模块都没有科学运算函数，不适合做矩阵等科学计算。因此，Numpy 没有使用 Python 本身的数组机制，而是提供了 ndarray 数组对象，该对象不但能方便地存取数组，而且拥有丰富的数组计算函数，比如向量的加法、减法、乘法等。

使用 ndarray 数组，首先需要导入 Numpy 函数库，可以直接导入该函数库（本节频繁使用 Numpy 库的函数，因此采用这种方法）。

```
from numpy import *
```

或者指定导入库的别名。

```
import numpy as np
```

下面正式进入 Numpy 的数组世界。如果没有说明，所称数组均为 Numpy 的数组对象，与 Python 的列表和 array 模块无关。

1) 创建数组。创建数组是进行数组计算的先决条件，可通过 array() 函数定义数组实例对象，其参数为 Python 的序列对象（比如列表）。如果想定义多维数组，则传递多层嵌套的

序列。例如下面这条语句定义了一个二维数组，其大小为 (2,3)，即共有 2 行，每行各 3 列。

```
a = np.array([[ 1., 7., 0.],[ -2., 1., 2.]])
```

上面语句定义了如表 3-1 所示的数组。

表 3-1 二维数组结构

1	7	0
-2	1	2

接着使用 array() 函数创建一个 (2,3) 大小的数组变量 x。

```
>>> from numpy import *
>>> x=array([[ 1., 0., 0.],[ 0., 1., 2.]])
```

以刚才定义的 x 变量为例，来熟悉 ndarray 数组对象的主要属性。ndarray 数组对象拥有 ndarray.ndim、ndarray.shape、ndarray.size、ndarray.dtype、ndarray.itemsize、ndarray.data 等属性。

ndarray.ndim：数组的维度数。

```
>>> x.ndim
2
```

ndarray.shape：数组的维数，返回的格式为 (n,m)，其中 n 为行数，m 为列数。

```
>>> x.shape
(2, 3)
```

ndarray.size：数组元素的总数。

```
>>> x.size
6
```

ndarray.dtype：数组元素的类型，比如：numpy.int32( 32 位整型)、numpy.int16( 16 位整型) 及 numpy.float64 ( 64 位浮点型)。

```
>>> x.dtype
dtype('float64')
```

ndarray.itemsize：数组中每个元素占有的字节大小。

```
>>> x.itemsize
8
```

ndarray.data：数组元素的缓冲区。

```
>>> x.data
<read-write buffer for 0x0557EAE8, size 48, offset 0 at 0x0561BAE0>
```

下面是一个关于 Numpy 的 ndarray 数组的例子，演示了 ndarray 数组的基本操作。

首先，创建 a 和 b 两个数组对象。其中，a 对象使用 Numpy 的 arange 函数产生了等差序



列数组 (Numpy 的 `arange` 函数与 Python 的 `range` 函数类似, 其参数依次为开始值、结束值、步长), 并用 `reshape` 函数创建了指定形状的新数组。a 的大小为 (3,5); b 的大小为 (1,3)。

```
>>> a = arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> b = array([6, 7, 8])
>>> b
array([6, 7, 8])
```

接着读取 a 和 b 的主要属性, 如: `shape`、`ndim`、`dtype`、`itemsize` 等。此外, 下面的代码还演示了 `type` 函数的使用, `type` 函数会返回对象的类型, 对于 `ndarray` 对象而言, 其类型为 `numpy.ndarray`。

```
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
>>> a.itemsize
4
>>> a.size
15
>>> type(a)
numpy.ndarray
>>> type(b)
numpy.ndarray
```

最后, 对 a 和 b 对象重新赋值, 以便进一步了解 `ndarray` 数组的元素类型。下面分别演示了 `int32`、`float64` 等类型的使用方法, 最后演示了不常见的复数作为数组元素 (数组 c 的元素) 的情况。

```
>>> a = array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int32')
>>> b = array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
>>> c = array([ [1,2], [3,4] ], dtype=complex )
>>> c
array([[ 1.+0.j,  2.+0.j],
       [ 3.+0.j,  4.+0.j]])
```

2) 特殊数组。Numpy 的特殊数组主要有以下几种:

❑ zeros 数组: 全零数组, 元素全为 0, 使用 `zeros` 函数创建。

□ ones 数组：全 1 数组，元素全为 1，使用 ones 函数创建。

□ empty 数组：空数组，元素全近似为 0，使用 empty 函数创建。

下面的代码依次演示了全零数组、全 1 数组、空数组的创建方法。

```
>>> zeros( (3,4) )
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
>>> ones( (2,3,4), dtype=int16 )
array([[[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]],
       [[ 1, 1, 1, 1],
        [ 1, 1, 1, 1],
        [ 1, 1, 1, 1]]], dtype=int16)
>>> empty( (5,3) )
array([[ 1.42185083e-299,  1.41906197e-299,  5.77420410e-300],
       [ 6.02082633e-300,  1.41971817e-299,  5.77379398e-300],
       [ 5.77440917e-300,  5.77386233e-300,  5.77440917e-300],
       [ 5.77386233e-300,  5.77440917e-300,  5.77386233e-300],
       [ 1.42130399e-299,  5.77440917e-300,  5.77406740e-300]])
```

3) 序列数组。刚才已经提到过 arange 函数，它与 Python 的 range 函数相似，但它属于 Numpy 函数库，其参数依次为开始值、结束值、步长。此外，还可使用 linspace 函数创建等差序列数组，其参数分别为起始值、终止值、元素数量。下面的代码分别演示了 arange 函数和 linspace 函数的用法。

```
>>> arange( 10, 30, 5 )
array([10, 15, 20, 25])
>>> arange( 0, 2, 0.3 )
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])
>>> linspace( 0, 2, 9 ) # 从 0 到 2, 9 个数字
array([ 0. , 0.25, 0.5 , 0.75, 1. , 1.25, 1.5 , 1.75, 2. ])
>>> x = linspace( 0, 2*pi, 100 ) # 从 0 到 2*pi 共 100 个数字
```

4) 输出数组。可使用 print 输出 Numpy 的数组对象。下面的代码是一维数组的创建和输出。

```
>>> a = arange(6) # 一维数组
>>> print a
[0 1 2 3 4 5]
```

下面的代码是二维数组的创建和输出，从输出结果可清晰地看出 b 的大小为 (4,3)。

```
>>> b = arange(12).reshape(4,3) # 二维数组
>>> print b
[[ 0 1 2]
 [ 3 4 5]
 [ 6 7 8]
 [ 9 10 11]]
```

5) 数组索引。Numpy 数组的每个元素、每行元素、每列元素都可以用索引访问, 不过要注意索引是从 0 开始的。比如, 某数组大小为 (2,3), 则第 2 行第 1 列元素的索引是 [1,0]。下面以三维数组为例, 演示数组的创建、输出及索引。

首先创建三维数组 c, 并输出其元素。

```
>>> c = arange(24).reshape(2,3,4)          # 三维数组
>>> print c
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

三维数组 c 可表示为如图 3-1 所示的形式。

[0,0,:]				[0,1,:]				[0,2,:]			
0	1	2	3	4	5	6	7	8	9	10	11
[1,0,:]				[1,1,:]				[1,2,:]			
12	13	14	15	16	17	18	19	20	21	22	23

图 3-1 三维数组 c

图 3-1 中单元格上方标注了 [0,0,:], [0,1,:] 等索引, 索引中的 “:” 表示该维度内的所有元素。对照图 3-1, 查找索引 [1,2,:] 处 (第 2 行第 3 列) 的所有元素和索引 [0,1,2] 处的元素, 可得出元素为 [20,21,22,23] 和 6。下面编写代码验证一下。

```
>>> print c[1,2,:]
[20 21 22 23]
>>> print c[0,1,2]
6
```

6) 数组运算。数组的加减乘除以及乘方运算方式为, 相应位置的元素分别进行计算。比如:

数组加法: `array([20,31,42,53])=array([20,30,40,50])+array([0,1,2,3])`

数组减法: `array([20,29,38,47])=array([20,30,40,50])-array([0,1,2,3])`

数组乘法: `array([[2,0],[0,4]])=array([[1,1],[0,1]])*array([[2,0],[3,4]])`

数组乘方: `array([0,1,2,3])` 的二次方 `=array([0,1,4,9])`

数组除法: `array([20.,15.,13.33333333,12.5])=array([20,30,40,50])/array([1,2,3,4])`

下面的代码演示了数组的加、减、乘、除及更多运算 (关键代码处注释了运算类型)。

```
>>> a = array([20,30,40,50])
>>> aa = arange(1, 5)
>>> a/aa### 除法
array([ 20.          ,  15.          ,  13.33333333,  12.5          ])
```

```

>>>
>>> b = arange( 4 )
>>> b
array([0, 1, 2, 3])
>>> c = a-b### 减法
>>> c
array([20, 29, 38, 47])
>>> b**2### 乘方
array([0, 1, 4, 9])
>>> 10*sin(a)### 数乘, sin 函数为正弦函数
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35### 指定条件判断后生成相应的布尔数组
array([True, True, False, False], dtype=bool)
>>> A = array( [[1,1],[0,1]] )
>>> B = array( [[2,0],[3,4]] )
>>> A*B### 乘法
array([[2, 0],
       [0, 4]])
>>> #dot 表示乘积。对一维数组计算的是点积, 对二维数组计算的是矩阵乘积
...# 此处表示矩阵乘积
... dot(A,B)
array([[5, 4],
       [3, 4]])
>>> a = ones((2,3), dtype=int)###ones 函数创建全 1 数组, 指定元素类型为 int
>>> b = random.random((2,3))### 创建随机数组, 指定大小为 (2,3)
>>> a *= 3### 数乘
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a### 加法
>>> b
array([[ 3.69092703,  3.8324276 ,  3.0114541 ],
       [ 3.18679111,  3.3039349 ,  3.37600289]])
>>> a += b ### 加法
>>> a
array([[6, 6, 6],
       [6, 6, 6]])
>>> a = random.random((2,3))
>>> a
array([[ 0.6903007 ,  0.39168346,  0.16524769],
       [ 0.48819875,  0.77188505,  0.94792155]])
>>> a.sum()### 求和
3.4552372100521485
>>> a.min()### 求最小值
0.16524768654743593
>>> a.max()### 求最大值
0.9479215542670073

```

7) 数组的拷贝。数组的拷贝分为浅拷贝和深拷贝两种, 浅拷贝通过数组变量的赋值完成, 深拷贝使用数组对象的 copy 方法。

浅拷贝只拷贝数组的引用, 如果对拷贝进行修改, 源数组也将修改。下面的代码演示了浅拷贝的方法。



```
>>> a=ones((2,3))
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b=a###b 为 a 的浅拷贝
>>> b[1,2]=2
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  2.]])
>>> b
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  2.]])
```

深拷贝会复制一份和源数组一样的数组，新数组与源数组不会存放在同一内存位置中，因此，对新数组的修改不会影响源数组。下面的代码演示了 b 使用 copy 方法从源数组 a 复制一份拷贝的情况。可以看到，修改 b 后，a 仍然不变。

```
>>> a=ones((2,3))
>>> b = a.copy()
>>> b[1,2]=2
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  2.]])
```

## 2. 矩阵

1) 创建矩阵。Numpy 的矩阵对象与数组对象相似，主要不同之处在于，矩阵对象的计算遵循矩阵数学运算规律。矩阵使用 matrix 函数创建，以 (2,2) 大小的矩阵 (2 行 2 列) 为例，可用以下两种方式定义参数：

'第 1 行第 1 列元素 第 1 行第 2 列元素；第 2 行第 1 列元素 第 2 行第 2 列元素'  
[[ 第 1 行第 1 列元素, 第 1 行第 2 列元素 ], [ 第 2 行第 1 列元素, 第 2 行第 2 列元素 ]]

下面的代码演示了矩阵的创建及类型查询方法。

```
>>> A = matrix('1.0 2.0; 3.0 4.0')### 矩阵 A
>>> A
[[ 1.  2.]
 [ 3.  4.]]
>>> B = matrix([[1.0,2.0],[3.0,4.0]])### 矩阵 B
>>> B
matrix([[ 1.,  2.],
        [ 3.,  4.]])
>>> type(A) # 查询 A 变量的类型
<class 'numpy.matrixlib.defmatrix.matrix'>
```

2) 矩阵运算。矩阵的常用数学运算有转置、乘法、求逆等。下面的代码演示了矩阵的基本运算 (请先导入 numpy 库再执行以下代码)。

```

>>> A.T           # 转置
[[ 1.  3.]
 [ 2.  4.]]
>>> X = matrix('5.0 7.0')
>>> Y = X.T       # 转置
>>> Y
[[5.]
 [7.]]
>>> print A*Y     # 矩阵乘法
[[19.]
 [43.]]
>>> print A.I     # 逆矩阵
[[-2.   1.]
 [ 1.5 -0.5]]
>>> solve(A, Y)   # 解线性方程组
matrix([[ -3.],
        [ 4.]])

```



注意 关于 Numpy 及其函数的更多信息可查阅 Numpy 官网：

[http://wiki.scipy.org/Numpy\\_Example\\_List](http://wiki.scipy.org/Numpy_Example_List)

### 3.1.3 pylab、matplotlib 绘图

为了验证算法的有效性，机器学习通常需要进行绘图，pylab、matplotlib 等模块是专业的 Python 绘图模块。

#### 1. sin 函数绘制

在二维坐标系中绘图的基本方式是使用 plot 方法，其参数分别为 x 轴数值、y 轴数值，这里的数值可以是单个数也可以是 Numpy 的一维数组对象。下面的代码演示了 sin 函数图像的绘制。

```

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 5, 0.1);
y = np.sin(x)
plt.plot(x, y)

```

如图 3-2 所示为 sin 函数图像效果图，从效果图上观察，曲线清晰，坐标系的标尺根据绘制参数已进行自动调整。

#### 2. cos 函数绘制

下面的代码使用 plot 方法绘制 cos 函数图像。

```

import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0, 5, 0.1);
y = np.cos(x)
plt.plot(x, y)
plt.show()

```

绘图效果如图 3-3 所示。

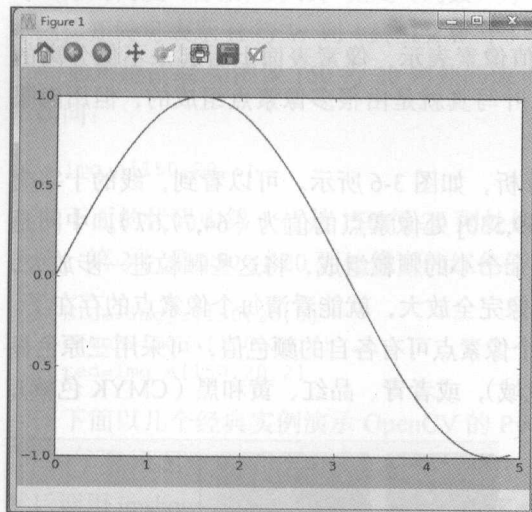


图 3-2 sin 函数图像效果图

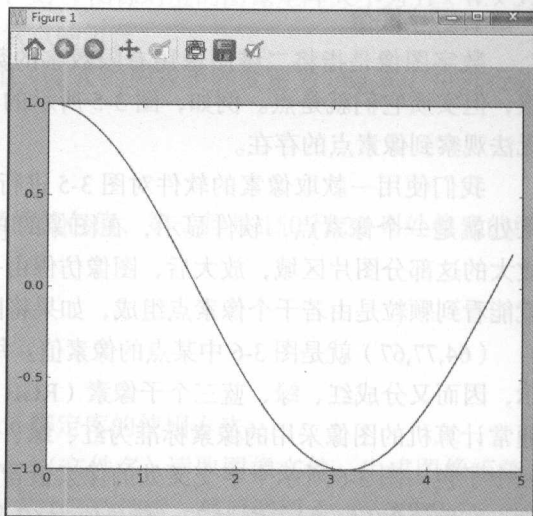


图 3-3 cos 函数图像效果图

进一步扩充图 3-3 所示 cos 函数绘制范围，将自变量  $x$  的范围扩大到  $[-8,8]$ ，三角函数 cos 图像的周期性一目了然，如图 3-4 所示。下面是绘制代码。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(-8, 8, 0.1);
y = np.cos(x)
plt.plot(x, y)
plt.show()
```

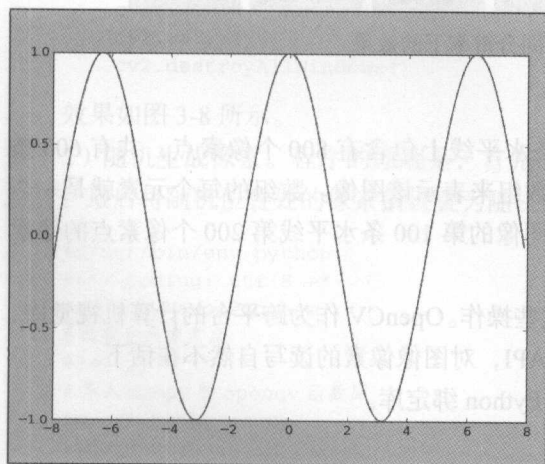


图 3-4 cos 函数周期图像

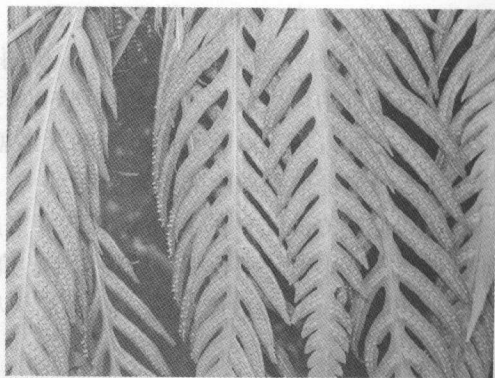


图 3-5 树叶写真

### 3.1.4 图像基础

#### 1. 数字图像

数字图像是指将二维图像用有限数字的数值像素表示，像素表面上看起来不像分离的点，但实质它们就是点。例如，图 3-5 所示的树叶写真就是由很多像素点组成的，但用肉眼无法观察到像素点的存在。

我们使用一款取像素的软件对图 3-5 进行分析，如图 3-6 所示。可以看到，线的十字交叉处就是一个像素点，软件显示，在图像的 [459,530] 处像素点的值为 (64,77,67)。中间是放大的这部分图片区域，放大后，图像仿佛由一个个小的颗粒组成，将这些颗粒进一步放大，就能看到颗粒是由若干个像素点组成，如果将图像完全放大，就能看清每个像素点的存在了。

(64,77,67) 就是图 3-6 中某点的像素值。每个像素点可有各自的颜色值，可采用三原色显示，因而又分成红、绿、蓝三个子像素 (RGB 色域)，或者青、品红、黄和黑 (CMYK 色域)，通常计算机的图像采用的像素标准为红、绿、蓝三个子色。图 3-6 所示十字交叉处的像素值含义为：红色值为 64，绿色值为 77，蓝色值为 67。

分辨率是度量图像内数据量多少的一个参数，通常表示成每英寸像素数和每英寸点数。分辨率越高，图像包含的数据越多，就越能表现更丰富的细节，图形文件就越大。从图 3-7 能较直观地看出这个效果，随着分辨率的增加，字母 R 越来越清晰。

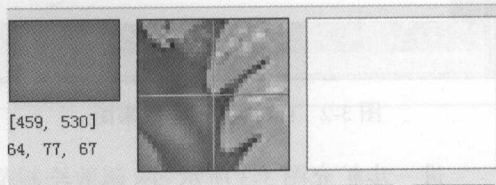


图 3-6 树叶放大的颗粒效果 (附彩图)

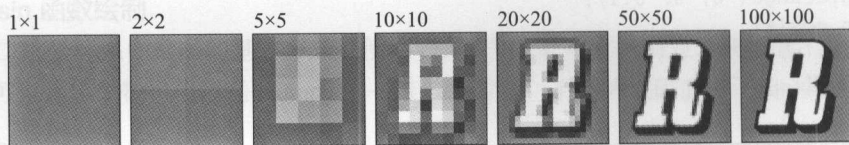


图 3-7 R 字母在不同分辨率下的效果

#### 2. OpenCV 的 Python 绑定库实例

假设图像的分辨率为  $800 \times 600$ ，则每一条水平线上包含有 800 个像素点，共有 600 条线，在计算机中可以使用一个 800 列 600 行的数组来表示该图像。数组的每个元素就是一个像素点，比如，第 100 行第 200 列的元素就是图像的第 100 条水平线第 200 个像素点的像素值。那么如何提取图像中的像素值呢？

可以使用 OpenCV 的 Python 绑定库完成这些操作。OpenCV 作为跨平台的计算机视觉库，拥有包括 500 多个跨平台图像处理的中、高层 API，对图像像素的读写自然不在话下。

使用 OpenCV 函数库之前，需要先导入其 Python 绑定库。

```
import cv2
```

OpenCV 函数对像素点的读写操作可理解为对图像矩阵的存取，OpenCV 图像矩阵中每



个像素点的值由蓝色值、绿色值、红色值3个部分组成，三色值组合成一个一维数组。假设A图像的高度(行数)为H，宽度(列数)为W，则A图像对应的图像矩阵大小为 $H \times W \times 3$ ，A图像矩阵可表示H行W列(共 $H \times W$ 个)像素点的组合。

如果想读取A图像150行20列处的像素值(设A的图像矩阵变量为

```
img_a[150,20,:]
```

下面的代码中第1行是150行20列处像素的蓝色值，第2行是150行20列处像素的绿色值，第3行是150行20列处像素的红色值。

```
blue=img_a[150,20,0]
green=img_a[150,20,1]
red=img_a[150,20,2]
```

下面以几个经典实例演示OpenCV的Python绑定库的使用方法。

1) 显示图像。程序原理是，首先使用

```
#!/usr/bin/env python
#3-1.py
import cv2
fn="test1.jpg"

if __name__ == '__main__':
    print 'http://blog.csdn.net/myhaspl'
    print 'myhaspl@qq.com'
    print
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    cv2.imshow('preview', img)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

效果如图3-8所示。

2) 随机生成像素。程序的原理是，首先产生空图像矩阵，然后确定矩阵的2000个随机位置，最后将随机位置处的像素值设置为随机数数组。下面是源代码。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
# 随机生成像素点
#3-2.py
# 导入 numpy 和 opencv 函数库
import numpy as np
import cv2
```

```
if __name__ == '__main__':
```

```

# 行数
sz1 = 200
# 列数
sz2 = 300
print u' 产生空图像矩阵 (%d*%d) ...' % (sz1, sz2)
# 产生空图像矩阵, 大小为 sz1*sz2 (行数 * 列数), 本程序为 200*300
img = np.zeros((sz1, sz2, 3), np.uint8)
pos1=np.random.randint(200,size=(2000, 1))### 行位置随机数组
pos2=np.random.randint(300,size=(2000, 1))### 列位置随机数组
# 在随机位置处设置像素点值
for i in range(2000):
    img[pos1[i],pos2[i],[0]]=np.random.randint(0,255)
    img[pos1[i],pos2[i],[1]]=np.random.randint(0,255)
    img[pos1[i],pos2[i],[2]]=np.random.randint(0,255)

# 显示图像
cv2.imshow('preview', img)
# 等待按键
cv2.waitKey()
# 销毁窗口
cv2.destroyAllWindows()

```



图 3-8 显示图像

随机产生像素点后, 创建新窗口并显示含彩色雪花点的图像, 运行以上代码:

产生空图像矩阵 (200\*300) ...

效果如图 3-9 所示。

3) 获取图像大小。程序通过图像矩阵的 shape 属性获取图像大小, shape 返回 tuple 元组, 元组的第 1 个元素为高度, 第 2 个元素为宽度, 第 3 个元素为 3 (像

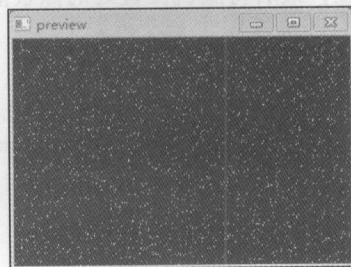


图 3-9 随机产生若干像素点 (附彩图)

素值由三原色组成)。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#3-3.py
import cv2
import numpy as np
fn="test2.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    # 获取图像矩阵大小
    sp=img.shape
    print sp
    # 高度, 即行数
    sz1=sp[0]
    # 宽度, 即列数
    sz2=sp[1]
    print 'width:%d\height:%d'%(sz2,sz1)
```

运行效果如下, 程序返回图像的高为 435, 宽为 656。

```
loading test1.jpg ...
(435, 656, 3)
width:656
height:435
```

4) 调节图像亮度。调节的原理是, 将像素值变小, 则将亮度调小, 全部色彩变暗; 将像素值变大, 则将亮度调大, 全部色彩变亮。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#3-4.py
import cv2
import numpy as np
fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print u'正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    ii=0
    # 将全部色彩变暗
    for xi in xrange(0,w):
        for xj in xrange(0,h):
            # 将像素值整体减少, 设为原像素值的 20%
            img[xj,xi,0]= int(img[xj,xi,0]*0.2)
            img[xj,xi,1]= int(img[xj,xi,1]*0.2)
            img[xj,xi,2]= int(img[xj,xi,2]*0.2)
        # 显示进度条
        if xi%10==0 :print '.',
    cv2.namedWindow('img')
```

```

cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()
print ''
print u'正在处理中',
# 将全部色彩变亮
for xi in xrange(0,w):
    for xj in xrange(0,h):
        # 将像素值整体增加, 设为原像素值的 1020%
        img[xj,xi,0]= int(img[xj,xi,0]*10.2)
        img[xj,xi,1]= int(img[xj,xi,1]*10.2)
        img[xj,xi,2]= int(img[xj,xi,2]*10.2)
    if xi%10==0 :print '.',
# 显示图像
cv2.namedWindow('img')
cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()

```

上面程序将图像每个像素值减少, 实现图像亮度变暗的效果, 如图 3-10 所示。然后每个像素值增大, 实现图像亮度变亮的效果。

如图 3-11 所示为图像变亮效果, 因为像素值过大, 已经出现失真现象。



图 3-10 图像变暗(附彩图)



图 3-11 图像变亮(附彩图)

5) 图像日落效果。日落效果的生成原理很简单, 将蓝色值和绿色值设为原来的 70%, 红色值不变, 设图像矩阵为 `img`。代码如下:

```

# 生成日落效果
for xi in xrange(0,w):
    for xj in xrange(0,h):
        img[xj,xi,0]= int(img[xj,xi,0]*0.7)### 蓝色值为原来的 70%
        img[xj,xi,1]= int(img[xj,xi,1]*0.7)### 绿色值为原来的 70%

```

完整代码如下:

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-

```



```

#3-5.py
import cv2
import numpy as np
fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print u'正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    ii=0
    # 生成日落效果
    for xi in xrange(0,w):
        for xj in xrange (0,h):
            img[xj,xi,0]= int(img[xj,xi,0]*0.7)### 蓝色值为原来的 70%
            img[xj,xi,1]= int(img[xj,xi,1]*0.7)### 绿色值为原来的 70%
        # 显示进度条
        if xi%10==0 :print '.',
    # 显示图像
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    cv2.waitKey()
    cv2.destroyAllWindows()

```

运行效果如图 3-12 所示。

6) 负片与水印效果。生成负片的原理是, 将像素的三色值设为 (255- 原值)。设图像矩阵为 `img`, 代码如下:

```

# 生成负片
b, g, r = cv2.split(img)
b=255-b
g=255-g
r=255-r

```

水印效果的原理是, 调用 `putText` 函数, 以图像矩阵为第 1 个参数, 输出内容为第 2 个参数, 在图像上直接输出水印文字。代码如下:

```

# 加上水印
cv2.putText(img,"machine learning", (20,20),cv2.FONT_HERSHEY_PLAIN, 2.0, (0, 0, 0), thickness = 2)
cv2.putText(img,"Support Vector Machines(SVMs)is an algorithm of machine learning.", (20,100),cv2.FONT_HERSHEY_PLAIN, 1.0, (0, 0, 0), thickness = 2)

```

负片与水印效果的完整代码如下:

```
#!/usr/bin/env python
```



图 3-12 图像日落效果 (附彩图)

```

#-*- coding: utf-8 -*-
#3-6.py
import cv2
import numpy as np
fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print u'正在处理中',
    # 读取图像文件
    img = cv2.imread(fn)
    # 获取图像大小
    w=img.shape[1]
    h=img.shape[0]
    ii=0
    # 生成负片
    b, g, r = cv2.split(img)
    b=255-b
    g=255-g
    r=255-r
    # 直接通过索引改变色彩分量
    img[:, :, 0]=b
    img[:, :, 1]=g
    img[:, :, 2]=r
    # 加上水印
    cv2.putText(img,"machine learning", (20,20),cv2.FONT_HERSHEY_PLAIN, 2.0, (0,
0, 0), thickness = 2)
    cv2.putText(img,"Support Vector Machines(SVMs)is an algorithm
of machine learning.", (20,100),cv2.FONT_HERSHEY_PLAIN, 1.0, (0, 0, 0),
thickness = 2)
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    cv2.waitKey()
    cv2.destroyAllWindows()

```

运行效果如图 3-13 所示。

7) 图像平铺。图像平铺的原理是, 首先计算平铺后的图像大小, 生成同样大小的空白图像, 然后在空白图像中逐个像素复制图像, 直接将空白图像像素值设置为平铺后该位置对应的像素值, 复制的顺序是逐行复制, 横向平铺 5 个图像, 纵向平铺 2 个图像, 最后显示图像效果。下面是完整代码:

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-7.py
import cv2

```

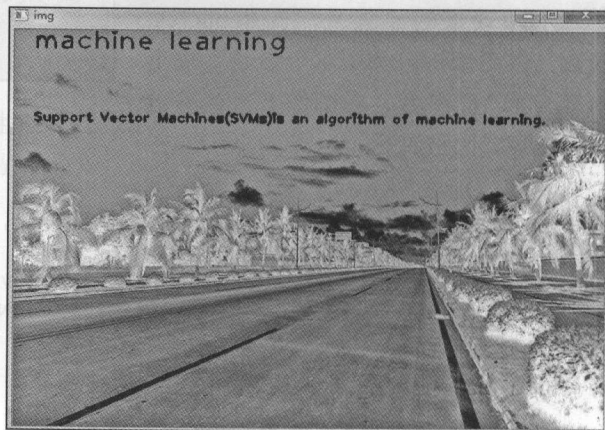


图 3-13 负片和水印效果 (附彩图)

```

import numpy as np
fn="test.png"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print '正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    # 横向平铺 5 个图像
    sz1=w*5
    # 纵向平铺 2 个图像
    sz0=h*2
    # 创建空白图像, 然后将图片排列
    myimg1=np.zeros((sz0,sz1,3), np.uint8)
    # 逐个像素复制
    img_x=0
    img_y=0
    for now_y in xrange(0,sz0):
        # 增加行数
        for now_x in xrange(0,sz1):
            # 复制对应位置的图像像素点
            myimg1[now_y,now_x,0]=img[img_y,img_x,0]
            myimg1[now_y,now_x,1]=img[img_y,img_x,1]
            myimg1[now_y,now_x,2]=img[img_y,img_x,2]
            # 增加列数
            img_x+=1
            if img_x>=w:
                img_x=0
                img_y+=1
            if img_y>=h:
                img_y=0
        print '.',
    cv2.namedWindow('img1')
    cv2.imshow('img1', myimg1)
    cv2.waitKey()
    cv2.destroyAllWindows()

```

运行效果如图 3-14 所示。

8) 转置并平铺图像。与刚才的例子相似, 但多了一步转置操作。转置的原理是将图像矩阵转换为它的转置矩阵, 转置算法是将新图像矩阵  $[h,w]$  处的像素设为原图像矩阵  $[w,h]$  处的值 (这里的值是一维矩阵), 相当于矩阵转置的算法。设  $myimg1$  为图像矩阵, 编写代码如下:

```

for now_y in xrange(0,sz0):
    for now_x in xrange(0,sz1):
        myimg1[now_x,now_y,:]=img[img_y,img_x,:]

```

转置并平铺图像完整代码如下:



图 3-14 图像平铺

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-8.py
import cv2
import numpy as np
fn="test.png"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print 'working',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    #w为宽度, h为高度
    sz1=w*2
    sz0=h*3
    # 创建空白图像, 然后将图片排列
    myimg1=np.zeros((sz1,sz0,3), np.uint8)
    # 翻转并生成图像
    # 逐个复制像素
    img_x=0
    img_y=0
    for now_y in xrange(0,sz0):
        for now_x in xrange(0,sz1):
            # 旋转图像
            myimg1[now_x,now_y,:]=img[img_y,img_x,:]
            img_x+=1
            # 新的一次平铺
            if img_x>=w:
                img_x=0
                img_y+=1
            # 新的一次平铺
            if img_y>=h:
                img_y=0
        print '.',
    cv2.namedWindow('img1')
    cv2.imshow('img1', myimg1)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

运行效果如图 3-15 所示。



图 3-15 图像旋转和平铺

### 3.1.5 图像融合与图像镜像

本节两个例子是对上节内容精华的总结, 较好地综合了 OpenCV 的基础功能。

#### 1. 图像融合

图像融合的原理是, 让新图像的每个像素成为两个源图像中对应像素的平均值之和, 即: 将两个图像的像素值取 50% 后相加。为简化计算, 直接选取其中一个源图像作为新图像, 设新图像矩阵为 myimg2, 编写代码如下:



```
# 每个像素为 2 个像素的平均值
for y in xrange(0,sz0):
    for x in xrange(0,sz1):
        myimg2[y,x,:]=myimg1[y,x,:]*0.5+myimg2[y,x,:]*0.5
```

完整代码如下（关键之处有注释）：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-9.py
import cv2
import numpy as np
fn1="he1.jpg"
fn2="he2.jpg"
if __name__ == '__main__':
    print 'working',
    myimg1 = cv2.imread(fn1)
    myimg2 = cv2.imread(fn2)
    # 取得图像大小
    w=myimg1.shape[1]
    h=myimg1.shape[0]
    sz1=w
    sz0=h

    # 每个像素为 2 个像素的 50% 之和，进行图像融合
    for y in xrange(0,sz0):
        for x in xrange(0,sz1):
            myimg2[y,x,:]=myimg1[y,x,:]*0.5+myimg2[y,x,:]*0.5
            print '.',

    cv2.namedWindow('img2')
    cv2.imshow('img2', myimg2)
    cv2.waitKey()
    cv2.destroyAllWindows()
```



效果如图 3-16 所示。

图 3-16 图像融合

## 2. 图像镜像

图像纵向镜像的原理是，首先获取图像的宽度，将宽度的 50% 取整后作为图像的纵向中线；然后以中线为轴，将图像左边反向复制到图像右边，使图像最右边一列的像素点等于图像最左边一列的像素点。比如，图像大小为  $200 \times 300$ （高 200，宽 300），第 180 行 170 列（索引为  $[180,170,:]$ ）的像素点值为第 180 行第 130 列的像素点值（ $300-170=130$ ）。

横向镜像与纵向镜像类似，不同之处在于将高度的 50% 取整后作为图像的横向中线，复制时是最下边一行的像素点值等于最上边一行的像素点值。

纵向镜像可按如下形式编写代码：

```
# 纵向生成镜像
mirror_w=w/2
for j in xrange(0,h):
```

```

        for i in xrange(0,mirror_w):
            img[j,i,:]=img[j,w-i-1,:]

```

下面的代码演示了图像的纵向镜像。

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-10.py
import cv2
import numpy as np

fn="test1.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    print '正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    ii=0
    # 纵向生成镜像
    mirror_w=w/2
    for j in xrange(0,h):
        for i in xrange (0,
mirror_w):
            img[j,i,:]=img[j,w-i
-1,:]

        print '.',
    # 显示图像
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    cv2.waitKey()
    cv2.destroyAllWindows()

```

运行效果如图 3-17 所示。



图 3-17 图像镜像

### 3.1.6 图像灰度化与图像加噪

#### 1. 图像灰度化

图像灰度化的原理是，彩色图像中的每个像素的颜色由 R、G、B 三个分量决定，而每个分量的取值范围为 0 ~ 255。而灰度图像是 R、G、B 三个分量相同的一种特殊的彩色图像，其算法有以下两种：

1) 求出每个像素点的 R、G、B 三个分量的平均值，然后将这个平均值赋予给这个像素的三个分量。

2) 根据 RGB 和 YUV 颜色空间的变化关系，建立亮度 Y 与 R、G、B 三个颜色分量的对应关系： $Y=0.3R+0.59G+0.11B$ ，以这个亮度值表达图像的灰度值。

OpenCV 有相关的函数 `cvtColor`，用它可直接完成灰度化操作。设 `img` 为源图像矩阵，`myimg1` 为灰度化后的目标图像矩阵，编写代码如下：

```
# 复制并转换为灰度化图像
myimg1=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

下面的代码演示了图像的复制与图像的灰度化操作。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#3-11.py
import cv2
import numpy as np

fn="test2.jpg"
if __name__ == '__main__':
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    sp=img.shape
    print sp
    # 获取图像大小
    #height
    sz1=sp[0]
    #width
    sz2=sp[1]
    # 显示图像大小
    print 'width:%d\nheight:%d'%(sz2,sz1)
    # 创建一个窗口并显示图像
    cv2.namedWindow('img')
    cv2.imshow('img', img)
    # 复制图像矩阵,生成与源图像一样的图像,并显示
    myimg2= img.copy();
    cv2.namedWindow('myimg2')
    cv2.imshow('myimg2', myimg2)

    # 复制并转换为灰度化图像,并显示
    myimg1=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    cv2.namedWindow('myimg1')
    cv2.imshow('myimg1', myimg1)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

上面的代码生成了与源图像一样的新图像,并生成了另一个源图像的灰度化图像,运行效果如图 3-18 所示。

现在大部分的彩色图像都是采用 RGB 颜色模式,处理图像的时候,要分别对 RGB 三种分量进行处理。实际上 RGB 并不能反映图像的形态特征,只是从光学的原理进行颜色的调配。把图像转换成 8 位的灰度值图像直接进行处理,可以通过直方图、灰

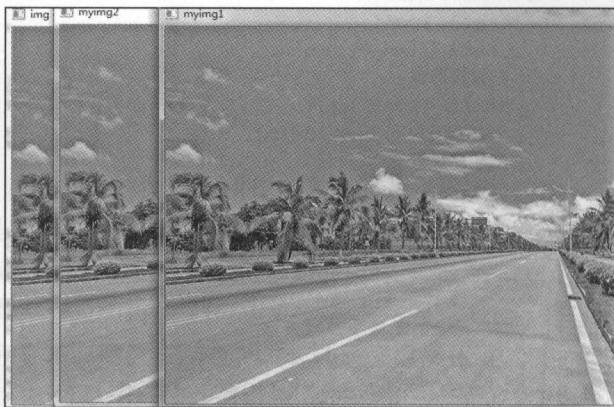


图 3-18 图像灰度化(附彩图)

度变化及正交变换之类数学运算对图像做进一步处理,比如说图像识别等。如果有必要,可将图像二值化,这样有利于对图像进一步处理,使图像数据量减小,突出感兴趣的目标的轮廓。如图 3-19 所示为某汽车图像二值化的效果。

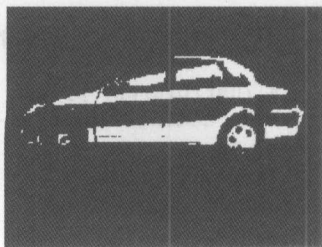


图 3-19 图像二值化

## 2. 图像加噪

给图像人为加噪的原理是,将图像若干个像素点的值设为噪声点的值。比如,为图像加上很多像素值为 [25,20,20] 的像素点,编写代码如下:

```
for k in xrange(0,coutn):
    xi = int(np.random.uniform(0,img.shape[1]))
    xj = int(np.random.uniform(0,img.shape[0]))
    if img.ndim == 2:
        # 灰度图像
        img[xj,xi] = 255
    elif img.ndim == 3:
        # 非灰度图像, 图像加噪
        img[xj,xi,0] = 25
        img[xj,xi,1] = 20
        img[xj,xi,2] = 20
```

上面的代码对 `img.ndim` 进行判断的用意在于,如果图像是灰度化图像,则 `img.ndim` 为 2,灰度化图像的像素值不存在红、绿、蓝三色之分,仅有灰度值,因此像素值仅需要一个,将对应噪声点位置的值设为 255 即可。

下面的代码演示了图像加噪的算法,为彩色图像人为加上 100 000 个色彩随机的噪声点。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#3-12.py
import cv2
import numpy as np
# 需要加噪的图像文件名
fn="test1.jpg"
if __name__ == '__main__':
    # 加载图像
    print 'loading %s ...' % fn
    img = cv2.imread(fn)
    # 噪声点数量
    coutn=100000
    for k in xrange(0,coutn):
        # 获取图像噪声点的随机位置
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        # 图像加噪声点
        if img.ndim == 2:
            img[xj,xi] = 255
        elif img.ndim == 3:
            img[xj,xi,0] = 25
```



```

img[xj,xi,1]= 20
img[xj,xi,2]= 20
cv2.namedWindow('img')
cv2.imshow('img', img)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行效果如图 3-20 所示。

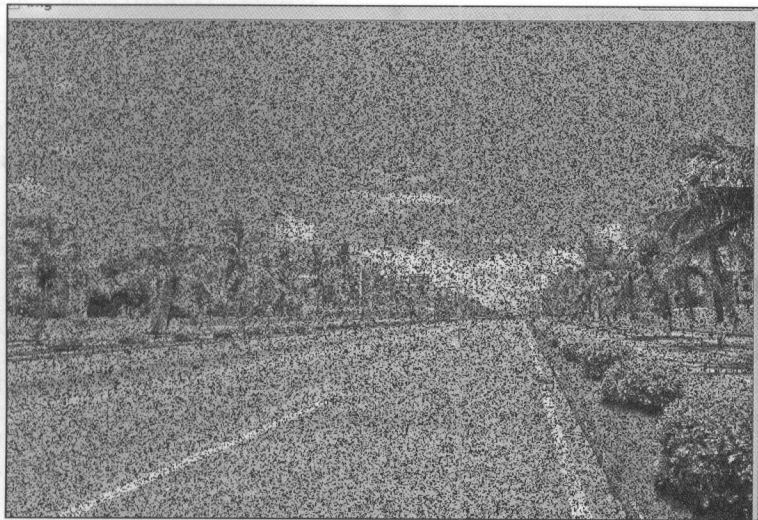


图 3-20 图像加噪

上述程序的运行原理是将图像数据矩阵随机位置的像素点设为 (25,20,20)，当随机的像素点数量较大时，就在图像上生成了噪声。

加上噪声的图像是为了实验图像识别的效果，有些机器学习算法对没有噪声的图像识别的效果很好，但如图 3-20 这种噪声较多的情况效果就很不理想了，因为在实际工程应用中，很难保证采集到的图像清晰可靠，所以需要人为给图像加上噪声，以方便后期对算法效果进行验证。

### 3.1.7 声音基础

#### 1. 声音原理

声音是由物体的机械振动形成的，发生声音的振动源叫作“声源”。振动着的鼓皮、琴弦、扬声器等都是声源，人的声带也是声源。声音必须通过媒质才能传播，空气、水、金属、木材等是最常见的媒质。声波的频率是每秒钟往复振动的次数，一来一往为一次，又称一周，声波的频率也就是声音的频率，频率单位为赫兹 (Hz)，每秒振动一周为 1Hz。“波长”是声源每振动一周声波所传播的距离，频率越高则波长越短，波长同频率成反比。

“相位”可简称为“相”。一般地说，相位是用来描述简谐振动的，在一个周波之内，任何一点的“相”都不相同，各对应于一个确定的相位角值；而在另一个周波，各种相位将会

重复出现。所以在声波传播的路径上,每隔一个波长的距离,其相位相同。

声音的音调是由它的基频决定的,基频越高则音调也越高。如在音乐中中央 C 的基频是 261.6Hz,而 A 调的基频则是 440Hz。通常将声音分为以下频带:20Hz、25Hz、31.5Hz、40Hz、50Hz、63Hz、80Hz、100Hz、125Hz、160Hz、200Hz、250Hz、315Hz、400Hz、500Hz、630Hz、800Hz、1kHz、1.25kHz、1.60kHz、2.0kHz、2.5kHz、3.15kHz、4.0kHz、5.0kHz、6.3kHz、8.0kHz、10kHz、12.5kHz、16kHz、20kHz。一般来说,人耳可感受的正弦波的范围是从 20Hz 的低频声音到 20kHz 的高频声。

## 2. 声音波形

声音波形属于正弦波,拥有振幅和频率两个特征,振幅就是音量,频率就是音调。下面调用 Python 的 WAV 声音处理库以及 Numpy 科学计算库显示一段声音的波形。

显示声音波形数据的主要步骤如下:

1) 打开 WAV 文件,使用 wave 库的 open 方法,主要参数为文件名和存取文件方式。

```
# 以读方式打开 WAV 文档
f = wave.open(r"back.wav", "rb")
```

2) 读取格式信息,使用 wave 库的 getparams 方法。该方法返回的信息中比较重要的是前 4 项,依次为通道数、样本宽度、样本频率、波形数据长度。

```
# 读取格式信息
# (nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
```

3) 读取波形数据,波形数据是 WAV 文件采样后生成的采样数据,使用 wave 库的 readframes 方法读取,该方法返回的数据是字符类型。

```
# 读取波形数据
str_data = f.readframes(nframes)
```

4) 转换波形数据为 Numpy 的整型数组对象。

```
# 将波形数据转换为数组
wave_data = np.fromstring(str_data, dtype=np.short)
wave_data.shape = -1, 2
wave_data = wave_data.T
```

5) 计算时间轴。

```
time = np.arange(0, nframes) * (1.0 / framerate)
```

6) 绘制波形,绘制前调用 pylab 的 subplot 方法创建两个上下形式的绘图区,每个绘图区各绘制一个声道的数据。下面程序中 subplot 方法的参数共 3 位整数,从左边开始每位依次表示绘图区总数、列数、创建区域所属绘图的索引,比如 subplot(212) 表示绘图区有 2 个,一共 1 列,当前索引为第 2 个绘图区。

```

# 绘制波形
# 创建上边的绘图区
pl.subplot(211)
# 绘制左声道
pl.plot(time, wave_data[0])
# 创建下边的绘图区
pl.subplot(212)
# 绘制右声道
pl.plot(time, wave_data[1], c="g")

```

上述绘制声音波形过程的完整代码如下：

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-13.py
import wave
import pylab as pl
import numpy as np
print 'working...'
# 打开 WAV 文档
f = wave.open(r"back.wav", "rb")
# 读取格式信息
#(nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
# 读取波形数据
str_data = f.readframes(nframes)
f.close()
# 将波形数据转换为数组
wave_data = np.fromstring(str_data, dtype=np.short)
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)
# 绘制波形
pl.subplot(211)
pl.plot(time, wave_data[0])
pl.subplot(212)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

```

程序读取声音文件后，绘制出如图 3-21 所示的波形。

这个波形表现出：声音信号较连续，随着时间的推移，变化不明显，没有停顿，因此，这是一段音乐或噪声等声音而不是人声，因为人说话的声音有个特点，就是每个字之间有少量停顿。语音停顿期间，声音采样软件采样不到数据，过了这个停顿期，波形会有明显的变化，如图 3-22 所示波形就是典型的说话声音波形。

### 3.1.8 声音音量调节

声音音量的调节方式与图像亮度调整类似，不同的是音量调节的是波形大小。音量调节

通过调节采样波形的大小实现, 采样数据变大时, 声音音量放大, 采样数据变小时, 声音音量降低。音量不能无限调节, 音量过大或过小, 会形成难听的噪音, 使声音失真。

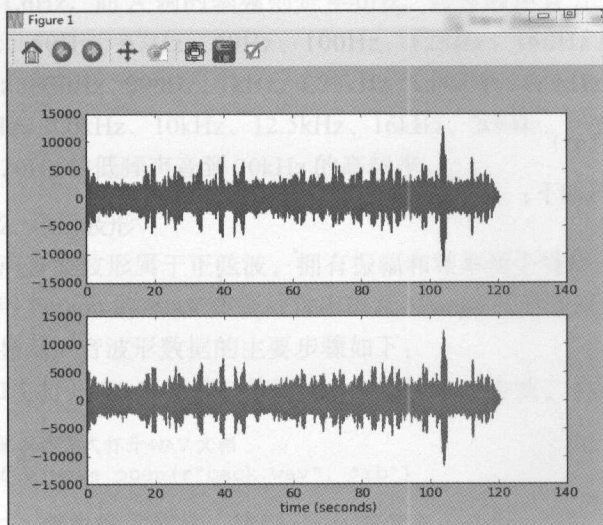


图 3-21 声音波形绘制

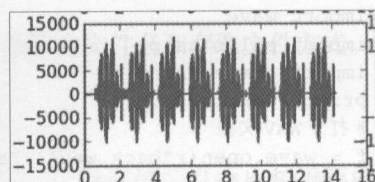


图 3-22 说话声音波形

### 1. 放大音量

下面编写代码演示音量的放大。为保证声音质量, 需要对音量调节范围设置上限和下限 (以原声音为基准计算上限、下限)。为此, 编写 `wavechange` 函数, 计算调整后的数据, 其参数 `x` 为每次采样的波形数据, `dwmax` 为上限, `dwmin` 为下限, 该函数仅会将上限与下限之间区域内的数据放大为原来的 1.5 倍, 在此区域外的数据则设置为上限或下限。

```
def wavechange(x, dwmax, dwmin):
    if x!=0:
        if abs(x)>dwmax:
            x=x/abs(x)*dwmax
        elif abs(x)<dwmin:
            x=x/abs(x)*dwmin
        else:
            x=x*1.5
    return x
```

为保证放大后声音不失真, 可采用以原声音为基准的放大策略, 声音波形图像类似正弦函数图像, 在以时间轴为 X 轴、采样数据为 Y 轴的坐标系中, 波形数据可正可负, 上下波动。因此, 以原声音数据的最大值为依据计算上下限, 上限为原声音数据最大值的 88%, 下限为原声音数据最大值的 14%。

使用 `wave_data.max()` 获取原声音波形的最大数据值 (`max` 函数返回数组的最大值), 然后通过 `frompyfunc` 函数设置调节音量的回调函数为刚刚定义的 `wavechange` 函数, 最后对数据进行放大调节。



# 放大音量

```
change_dwmax=wave_data.max()/100*88
```

```
change_dwmin=wave_data.min()/100*14
```

```
wave_change = np.frompyfunc(wavechange,3,1)
```

```
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
```

声音数据放大后, 需要将新数据写入新的声音文件中。首先以写方式新建新声音文件:

```
fo = wave.open(r"jg.wav", "wb")
```

然后, 设置新文件的数据参数为源文件的数据参数, 并写到新声音文件中。放大音量并没有改变格式信息, 因此, 放大后的声音与源声音的格式信息一样。

# 写波形数据参数

```
print "save new wav files...."
```

```
fo.setnchannels(nchannels)
```

```
fo.setframerate(framerate)
```

```
fo.setsampwidth(sampwidth)
```

最后调用 `writframes()` 方法, 以放大后声音数据为参数将数据写入新建的声音文件中。

```
fo.writeframes(new_str_data)
```

下面的代码演示了放大音量的算法, 并绘制出了源声音波形与放大后的声音波形。

```
#!/usr/bin/env python
```

```
# -*- coding: utf-8 -*-
```

```
#code:myhaspl@qq.com
```

```
#3-14.py
```

```
import wave
```

```
import pylab as pl
```

```
import numpy as np
```

```
def wavechange(x,dwmax,dwmin):
```

```
    if x!=0:
```

```
        if abs(x)>dwmax:
```

```
            x=x/abs(x)*dwmax
```

```
        elif abs(x)<dwmin:
```

```
            x=x/abs(x)*dwmin
```

```
        else:
```

```
            x=x*1.5
```

```
    return x
```

```
# 打开 WAV 文档
```

```
f = wave.open(r"speak.wav", "rb")
```

```
fo = wave.open(r"jg.wav", "wb")
```

```
# 读取波形数据
```

```
params = f.getparams()
```

```
nchannels = f.getnchannels()
```

```
nchannels, sampwidth, framerate, nframes = params[:4]
```

```
print "read wav data...."
```

```
str_data = f.readframes(nframes)
```

```
# 将波形数据转换为数组, 并更改
```

```
print "update wav data...."
```

```
wave_data = np.fromstring(str_data, dtype=np.short)
```

```
params = f.getparams()
```

```

nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
# 放大音量
change_dwmax=wave_data.max()/100*88
change_dwmin=wave_data.min()/100*14
wave_change = np.frompyfunc(wavechange,3,1)
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
# 写波形数据参数
print "save new wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)
# 绘制源声音波形
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(221)
pl.plot(time, wave_data[0])
pl.subplot(222)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
# 绘制放大音量波形
new_wave_data.shape = -1, 2
new_wave_data =new_wave_data.T
new_time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(223)
pl.plot(new_time,new_wave_data[0])
pl.subplot(224)
pl.plot(new_time, new_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

```

如图 3-23 所示波形图中，上部为源声音，下部显示了音量放大后的波形。下部的波形数据范围为  $-20\,000 \sim 20\,000$ ，而上部范围为  $-15\,000 \sim 15\,000$ ，下部波形整体比上部大很多。下载本书的代码包运行后，可听到音量放大后的声音效果。

## 2. 降低音量

音量降低可通过将采样波形变小来实现，具体来说，就是把每个采样数据按指定比例缩小，同时将缩小幅度控制在合理的范围内，保证音量降低后声音

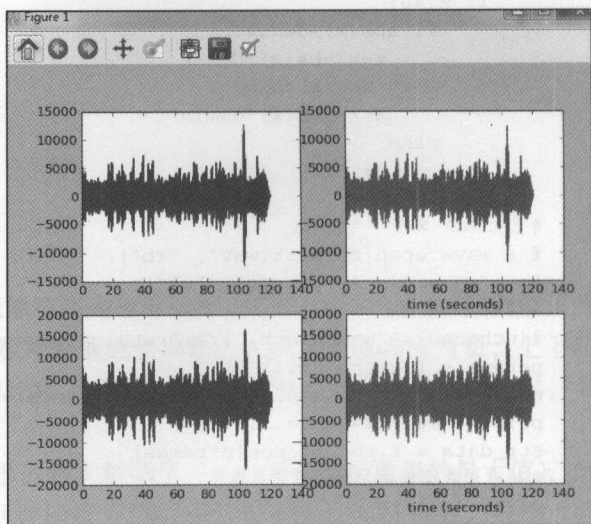


图 3-23 声音音量放大波形

仍然清晰。

1) 根据上下限参数对波形数据进行调节, 定义缩小波形数据的函数为 `wavechange`。

```
def wavechange(x, dwmax, dwmin):
    if x!=0:
        if abs(x)<dwmax and abs(x)>dwmin:
            x=x*0.5
        else:
            x=x*0.2
    return x
```

2) 与放大音量类似, 以源声音波形数据的最大值为基准, 计算上限和下限, 以 `wavechange` 为回调函数, 降低音量。

```
# 降低音量
change_dwmax=wave_data.max()/100*1
change_dwmin=wave_data.max()/100*0.5
wave_change = np.frompyfunc(wavechange,3,1)
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
```

3) 生成新波形数据。

```
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
```

4) 将数据写到新声音文件。

```
# 写波形数据参数
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)
```

下面的代码演示了降低音量算法。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-15.py
import wave
import pylab as pl
import numpy as np
print 'working...'
def wavechange(x, dwmax, dwmin):
    if x!=0:
        if abs(x)<dwmax and abs(x)>dwmin:
            x=x*0.5
        else:
            x=x*0.2
    return x
# 打开WAV文档
f = wave.open(r"back.wav", "rb")
fo = wave.open(r"jg.wav", "wb")
# 读取波形数据
```

```

#(nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
print "read wav data...."
str_data = f.readframes(nframes)
# 将波形数据转换为数组, 并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)
# 降低音量
change_dwmax=wave_data.max()/100*1
change_dwmin=wave_data.min()/100*0.5
wave_change = np.frompyfunc(wavechange,3,1)
new_wave_data =wave_change(wave_data,change_dwmax,change_dwmin)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
# 写波形数据参数
print "save new wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)
# 绘制源声音波形
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(221)
pl.plot(time, wave_data[0])
pl.subplot(222)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
# 绘制降低音量波形
new_wave_data.shape = -1, 2
new_wave_data =new_wave_data.T
new_time = np.arange(0, nframes) * (1.0 / framerate)
pl.subplot(223)
pl.plot(new_time,new_wave_data[0])
pl.subplot(224)
pl.plot(new_time, new_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()

```

如图 3-24 所示的波形图中, 上面为源声音, 下面为降低音量后的声音波形。可明显看出, 音量缩小后, 其波形幅度为  $-3000 \sim 3000$ , 而源声音波形范围大很多, 为  $-15\,000 \sim 15\,000$ 。

### 3.1.9 图像信息隐藏

#### 1. 图像隐藏原理

信息隐藏是不让除预期接收者之外的任何人知晓信息的传递事件或者信息的内容, 载体



文件相对隐秘文件的大小越大,隐藏后者就越加容易。因此,数字图像在互联网和其他传媒上被广泛用于隐藏消息。

本节讲述的图像隐藏原理是:首先从源图中提取文字图像信息,并记录这个文字图像信息像素点在图像矩阵中的位置;然后,对载体文件进行预处理,将蓝色像素值全部设为偶数;最后,将记录的文字信息像素点在载体文件对应位置的蓝色像素值设为奇数。解密信息是隐藏信息的逆过程,其过程比较简单,即提取载体文件中蓝色像素值为奇数的像素点,将空白图像中这些像素点对应的位置赋予统一的着色。

## 2. 图像隐藏实例

下面用实例来讲解图像信息隐藏技术。我们的目标是:将如图 3-25 所示的文字隐藏在如图 3-26 所示的载体图片里。要求隐藏后,无法察觉图中隐藏了信息。

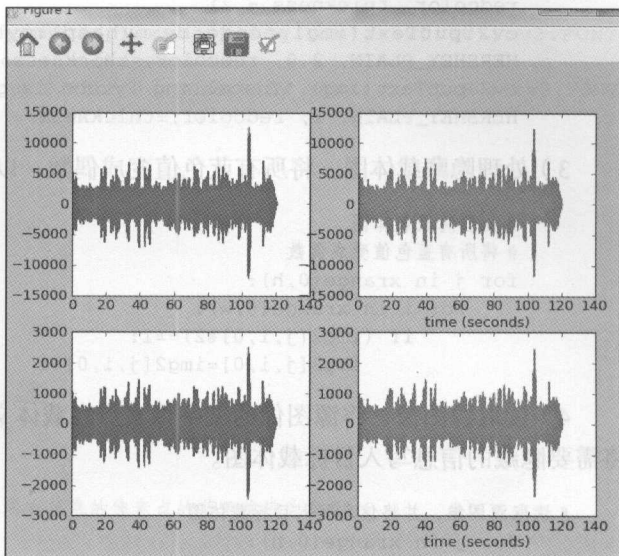


图 3-24 声音音量缩小波形



图 3-25 含有待隐藏文字的图像



图 3-26 载体图像

本实例隐藏信息的主要过程如下:

- 1) 读取源图像(将写上需隐藏文字的信息)和载体图像,构造图像矩阵。

```
img1 = cv2.imread(fn1)
img2 = cv2.imread(fn2)
w=img1.shape[1]
h=img1.shape[0]
```

- 2) 在源图像中加上水印文字作为待隐藏文字。

# 加上需要隐藏的消息

```
cv2.putText(img1,"hello,world!", (20,300),cv2.FONT_HERSHEY_PLAIN, 3.0,
redcolor, thickness = 2)
cv2.putText(img1,"code by myhaspl:myhaspl@qq.com", (20,60),cv2.FONT_
HERSHEY_PLAIN, 2.0, redcolor, thickness = 2)
cv2.putText(img1,"Installing Python is generally easy. ", (1,90),cv2.FONT_
HERSHEY_PLAIN, 2, redcolor, thickness = 1)
```

3) 处理隐藏载体图, 将所有蓝色值变成偶数, 以便加入隐藏信息。

```
# 处理隐藏载体图
# 将所有蓝色值变成偶数
for j in xrange(0,h):
    for i in xrange(0,w):
        if (img2[j,i,0]%2)==1:
            img2[j,i,0]=img2[j,i,0]-1
```

4) 读取源图像, 将源图像的文字像素点在载体文件的对应位置的蓝色像素值设为奇数, 将需要隐藏的信息写入目标载体图。

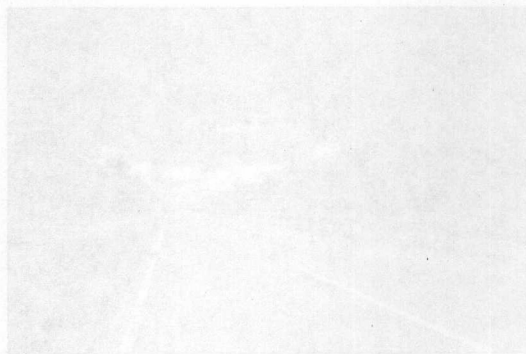
```
# 读取源图像, 并将信息写入目标载体图
for j in xrange(0,h):
    for i in xrange(0,w):
        if (img1[j,i,0],img1[j,i,1],img1[j,i,2])==redcolor:
            img2[j,i,0]=img2[j,i,0]+1
```

5) 保存修改后的目标载体图。

```
cv2.imshow('img2-2', img2)
cv2.imwrite(fn3, img2)
```

下面的代码演示了隐藏信息的过程。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#3-16.py
import cv2
import numpy as np
# 含有文字的图像
fn1="test1.jpg"
# 载体文件
fn2="test2.jpg"
# 包含隐藏信息的载体文件
fn3="secret.png"
redcolor=(0, 0, 255)
if __name__ == '__main__':
    print u'正在处理中',
    # 图像大小
    img1 = cv2.imread(fn1)
    img2 = cv2.imread(fn2)
    w=img1.shape[1]
    h=img1.shape[0]
```



```

# 加上需要隐藏的消息
cv2.putText(img1, "hello, world!", (20, 300), cv2.FONT_HERSHEY_PLAIN, 3.0,
redcolor, thickness = 2)
cv2.putText(img1, "code by myhaspl:myhaspl@qq.com", (20, 60), cv2.FONT_
HERSHEY_PLAIN, 2.0, redcolor, thickness = 2)
cv2.putText(img1, "Installing Python is generally easy. ", (1, 90), cv2.FONT_
HERSHEY_PLAIN, 2, redcolor, thickness = 1)

cv2.namedWindow('img1')
cv2.imshow('img1', img1)
cv2.namedWindow('img2-1')
cv2.imshow('img2-1', img2)
# 处理隐藏载体图
# 将所有蓝色值变成偶数
for j in xrange(0, h):
    for i in xrange(0, w):
        if (img2[j, i, 0] % 2) == 1:
            img2[j, i, 0] = img2[j, i, 0] - 1
        print '.',
        mirror_w = w / 2
# 读取源图, 并将信息写入目标图, 将有信息的像素点的蓝色值设为奇数
for j in xrange(0, h):
    for i in xrange(0, w):
        if (img1[j, i, 0], img1[j, i, 1], img1[j, i, 2]) == redcolor:
            img2[j, i, 0] = img2[j, i, 0] + 1
        print '.',
# 保存修改后的目标图, 并显示
cv2.namedWindow('img2-2')
cv2.imshow('img2-2', img2)
cv2.imwrite(fn3, img2)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行上段代码将信息隐藏后, 肉眼观察载体图像, 仍无法察觉与之前相比有任何变化。下面来看看解密信息过程。解密信息与隐藏信息相反, 是隐藏信息的逆过程, 主要步骤如下:

#### 1) 读取载体文件及其大小信息。

```

img = cv2.imread(fn)
w = img.shape[1]
h = img.shape[0]

```

#### 2) 生成空白图像矩阵, 以便绘制解密文字。

```

imginfo = np.zeros((h, w, 3), np.uint8)

```

#### 3) 绘制解密的水印文字。如果蓝色值为奇数, 则该像素点为文字。

```

for j in xrange(0, h):
    for i in xrange(0, w):
        if (img[j, i, 0] % 2) == 1:
            imginfo[j, i, 1] = 255

```

#### 4) 显示隐藏信息。

```
cv2.imshow('info', imginfo)
cv2.imwrite(fn, imginfo)
```

下面代码演示了解密信息的过程。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
# 解密文件
#3-17.py
import cv2
import numpy as np
fn="secret.png"
if __name__ == '__main__':
    print 'loading ...'
    print u'正在处理中',
    img = cv2.imread(fn)
    w=img.shape[1]
    h=img.shape[0]
    imginfo =np.zeros((h,w,3), np.uint8)
    for j in xrange(0,h):
        for i in xrange(0,w):
            if (img[j,i,0]%2)==1:
                # 如果蓝色值为奇数, 则该像素点为文字
                imginfo[j,i,1]=255
            print '.',
    cv2.imshow('info', imginfo)
    cv2.imwrite(fn, imginfo)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

运行解密代码, 从载体文件中提取信息, 效果如图 3-27 所示。

### 3.1.10 声音信息隐藏

#### 1. 声音信息隐藏原理

声音文件是一个不错的信息隐藏载体, 声音文件数据量大, 能隐藏信息的容量也大, 假设每秒采集 44 100 次, 如果所有采样数据全部利用上, 每秒的声音可以存储 44 100 字节的数据, 不过这样达不到信息隐藏的效果, 只能利用其中一部分采样数据来存储信息, 占有的采样数据越少, 信息隐藏效果就越好。

比如, 如图 3-28 所示的波形是一段音乐的声音波形, 假设某个采样点的数据实际是信息中一个字节大小的数据, 那么将这些字节解密后, 能还原成一段信息。这种载体的隐藏信息的效果比图像好, 一般很难被人发现。

这里采用的隐藏策略是: 产生一段正弦波的噪声, 然后, 在这段噪声中隐藏一段文本文件的内容。下面以实例来讲解这个过程, 我们的目标是: 将本章前面讲述的 Python 代码文件 3-1.py 隐藏到一段噪声中, 解密者如果不知道信息解密的规律, 就无法从噪声文件还原这个 Python 代码文件。



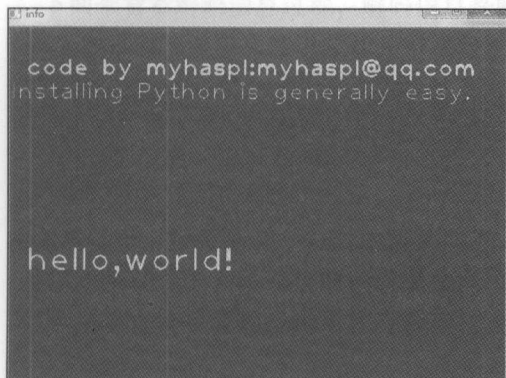


图 3-27 解密后的文字

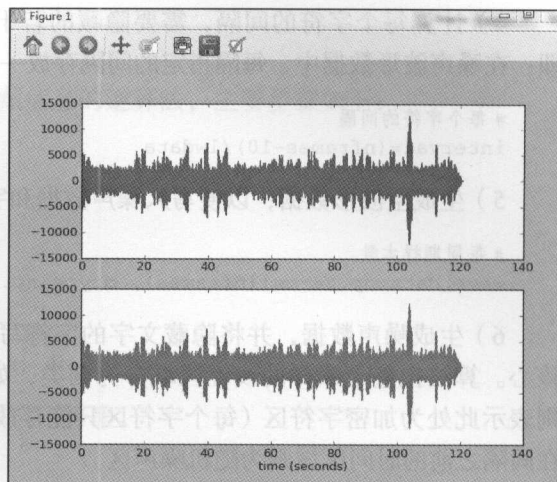


图 3-28 音乐的声音波形

## 2. 声音信息隐藏实例

隐藏信息的具体过程如下：

1) 读取需要隐藏的文本文件，提取其中的文字信息。

```
# 打开文档
fo = wave.open(r"pltest.wav", "wb")
file_object = open('3-1.py')
try:
    all_the_text = file_object.read()
finally:
    file_object.close()
```

2) 将文字转化为对应的内部编码（本例的文字为英文字母和符号，因此转换为 ASCII 码）。

```
wdata=map(ord,all_the_text)
wdata=np.array(wdata)
```

3) 设置噪声载体文件的波形参数。载体文件是程序人为生成，所以将幅度设置为适合的区域，为使载体噪声更接近于自然的噪声，将振幅范围设置为  $-25\ 600 \sim 25\ 600$ 。

```
# 设置波形参数
# 采样率
framerate = 44100
# 声道数
nchannels=2
# 每位宽度
sampwidth=2
# 长度
nframes =framerate*4
# 振幅
base_amplitude = 200
max_amplitude=128*base_amplitude
```

4) 计算每个字符的间隔, 需要隐藏的若干个字符以等间隔的形式分散在噪声数据中, 即: 在噪声波形数据中, 每隔指定的间隔存放一个字符。

```
# 每个字符的间隔
interval=(nframes-10)/lwdata
```

5) 生成空波形数据, 以便写入噪声数据和字符信息。

```
# 每周周期样本数
wave_data=np.zeros((nframes), dtype=np.short)
```

6) 生成噪声数据, 并将隐藏文字的字符写入噪声数据中, 这一步是关键, 也是算法的核心。算法把整个采样的线性区域分为两类, 如果当前采样时间处于第 4 步计算的间隔处, 则表示此处为加密字符区 (每个字符区只能存放一个字符), 写入的数据为经过加密的字符, 在间隔之前的时间区域则为随机噪声区。

算法判断是否到了指定的间隔处, 间隔处是字符区, 否则是噪声区。如果是噪声区, 则随机生成噪声; 如果是字符区, 则将字符进行加密, 写入字符区。此处使用了简单的加密算法 (实际应用可使用高强度的加密算法), 加密方式为: 将字符的 ASCII 码乘以指定的整数后, 减去 64 与该整数的乘积。生成的信息隐藏格式如表 3-2 所示。

表 3-2 信息隐藏格式

区域	随机噪声区	加密字符区	随机噪声区	加密字符区	.....
长度	interval	1	interval	1	

算法代码如下:

```
# 写噪声数据和隐藏文字的字符
myrand=np.random.rand(nframes)
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        # 将隐藏文字的字符通过一定的变化写入噪声数据中
        possamp=wdata[count]*base_amplitude-64*base_amplitude
        count+=1
    elif curpos%60==0:
        # 生成随机噪声数据
        possamp=int(myrand[curpos]*max_amplitude-max_amplitude/2)
    else:
        possamp=0
    wave_data[curpos]=possamp
```

7) 写波形数据。

```
# 写波形数据参数
print "save new wav files...."
str_data=wave_data.tostring()
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
```

```
fo.setnframes(nframes)
fo.writeframes(str_data)
```

下面来看看解码信息过程，解码信息是隐藏信息的逆算法，主要步骤如下：

### 1) 读取噪声载体文件以及相关格式信息。

```
new_wdata=[]
print u' 正在从声音解码文件 '
fi = wave.open(r"pltest.wav", "rb")
fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)
```

### 2) 找到字符区，将其中的字符解密并还原成字符串。

```
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        possamp=(fi_wave_data[curpos]+64*base_amplitude)/base_amplitude
        new_wdata.append(possamp)
        count+=1
```

### 3) 整理还原字符串，将它们写入文件。

```
my_the_text="".join(map(chr,new_wdata))
fmy_the_text="".join(map(chr,new_wdata))
file_object = open('mytext.txt', 'w')
file_object.write(my_the_text)
file_object.close()
```

下面程序读取名为 3-1.py 的 Python 源程序文件，将该文本隐藏在声音文件中，然后打开载体声音文件，将文本还原为 Python 程序文件。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
# 将文件隐藏在声音之中
#3-18.py
import wave
import pylab as pl
import numpy as np
# 编码
print u' 正在将文件编码进声音 '
print "generate wav data...."
# 打开文档
fo = wave.open(r"pltest.wav", "wb")
file_object = open('3-1.py')
try:
    all_the_text = file_object.read()
finally:
    file_object.close()
wdata=map(ord,all_the_text)
```

```

wdata=np.array(wdata)
lwdata=len(wdata)
# 设置波形参数
# 采样率
framerate = 44100
# 声道数
nchannels=2
# 每位宽度
sampwidth=2
# 长度
nframes =framerate*4
# 振幅
base_amplitude = 200
max_amplitude=128*base_amplitude
# 每个字符的间隔
interval=(nframes-10)/lwdata
# 每周取样本数
wave_data=np.zeros((nframes), dtype=np.short)
count=0
# 写噪声数据和隐藏文字的字符
myrand=np.random.rand(nframes)
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        possamp=wdata[count]*base_amplitude-64*base_amplitude
        count+=1
    elif curpos%60==0:
        possamp=int(myrand[curpos]*max_amplitude-max_amplitude/2)
    else:
        possamp=0
    wave_data[curpos]=possamp
# 写波形数据参数
print "save new wav files...."
str_data=wave_data.tostring()
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.setnframes(nframes)
fo.writeframes(str_data)
fo.close()
# 绘制波形
wave_data.shape = -1, 2
wave_data = wave_data.T
time = np.arange(0, nframes/2)
pl.subplot(211)
pl.plot(time, wave_data[0], c="r")
pl.subplot(212)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
# 解码
new_wdata=[]
print u' 正在从声音解码文件 '
fi = wave.open(r"pltest.wav", "rb")

```



```

fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)
count=0
for curpos in xrange(0,nframes):
    if curpos % interval==0 and count<lwdata:
        possamp=(fi_wave_data[curpos]+64*base_amplitude)/base_amplitude
        new_wdata.append(possamp)
        count+=1
my_the_text="".join(map(chr,new_wdata))
file_object = open('mytext.txt', 'w')
file_object.write(my_the_text)
file_object.close()
pl.show()

```

运行这段代码，程序输出了编码和解码过程。

正在将文件编码进声音  
generate wav data....  
save new wav files....  
正在从声音解码文件

上面程序演示了隐藏信息与解码信息的过程。程序运行后，先将 Python 源代码文件 3-1.py 隐藏在一段噪声中。然后，从噪声中解码信息，将文本内容输出到 mytext.txt 中，用记事本打开该文件，如图 3-29 所示。可以看到解码后的程序和源程序一样，包括空格、符号及字母等。

如图 3-30 所示是程序运行后生成的声音波形图像，很难看出来哪些采样点隐藏了文本信息。下载本书源码包后，可以播放这段声音（运行 3-18.py 后，会产生声音文件 pltest.wav），只能听出声音是一段很平常的噪声。

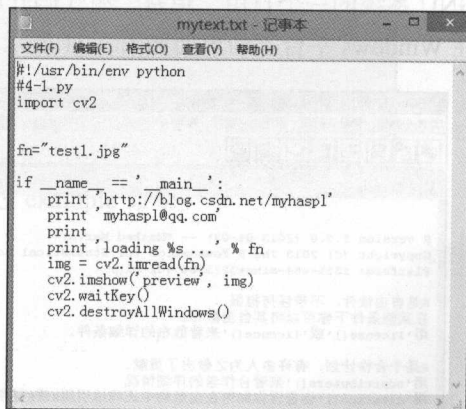


图 3-29 解码后文件

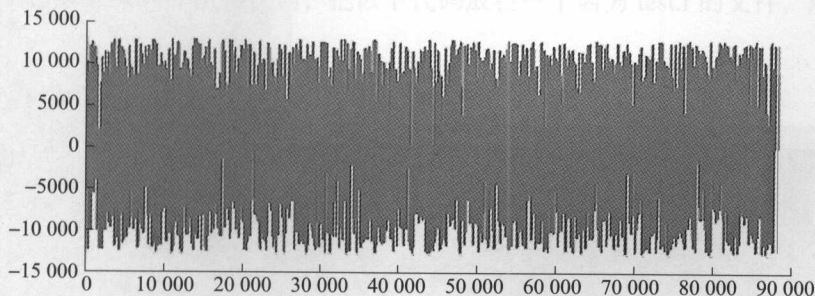


图 3-30 隐藏了文本信息的波形

### 3. 信息隐藏与解码总结

综上所述，隐藏信息的算法过程为：

1) 读取程序文本文件，将字符转换为 ASCII 码。

2) 确定声音文件的相关参数，生成 2 秒声音，其中采样数据用随机数代替，生成随机数的取值范围为  $-15\ 000 \sim 15\ 000$ ，在某些采样点上用来自隐藏信息的字符字节代替，代替的方式是将字节对应的 ASCII 码乘上某个基数加一个调整参数，代替的过程是线性的。

3) 将生成的声音数据写入载体文件中。

解码信息的算法过程为：

1) 读取载体声音文件及其相关参数。

2) 按照隐藏信息时的规律，在正确的位置读取字符，然后将读取的字符合成信息。

3) 将信息写入恢复文件中。

## 3.2 R 语言基础

R 是用于统计分析、绘图的语言和操作环境，是统计计算和统计制图的优秀工具，属于 GNU 系统的一个自由、免费、源代码开放的软件。其 GUI 界面主要包括菜单、命令控制台，在 Windows 平台下的界面如图 3-31 所示。

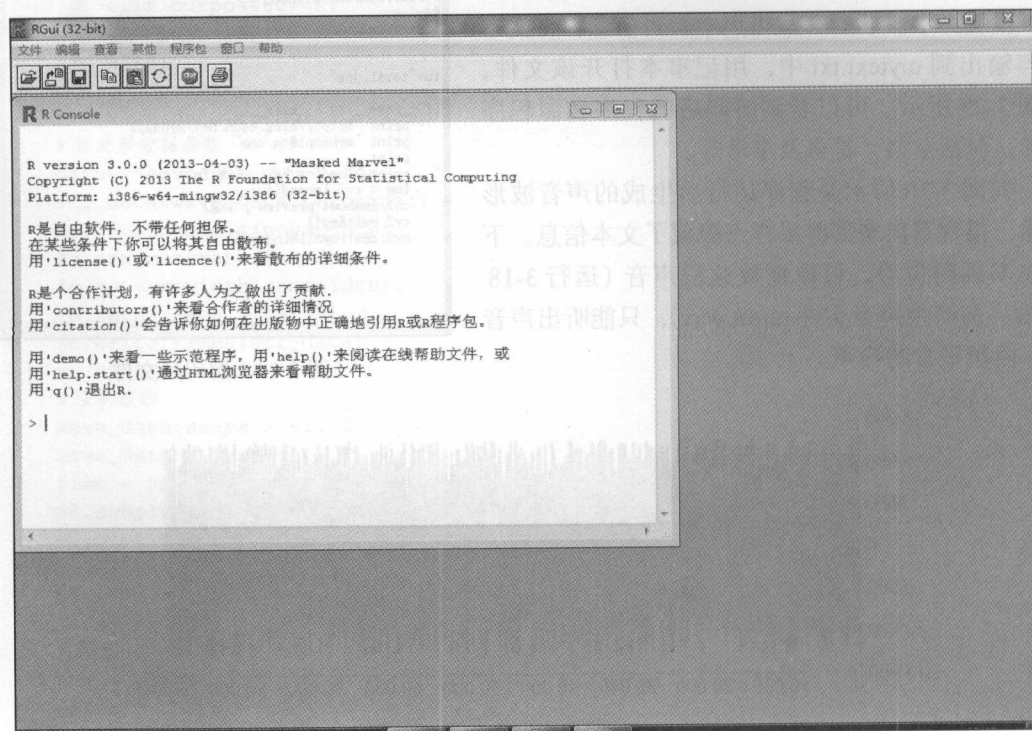


图 3-31 R 的 GUI 界面

### 3.2.1 基本操作

#### 1. 提示符

R 以 “>” 为 shell 提示符，Windows、Linux、MAC 均一致。

#### 2. 获得帮助的方式

在 R 中使用 `help` 函数获取某个命令或函数的帮助。下面是获取求平均值函数的帮助函数：

```
> help(mean)
starting httpd help server ... done>
```

输入上述命令后，显示如下 HTML 形式的帮助文档：

```
mean {base}
```

#### Arithmetic Mean

#### Description

Generic function for the (trimmed) arithmetic mean.

#### Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

如果想进一步获得这个函数的调用示例，可以通过 `example` 命令。

```
> example(mean)
mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

#### 3. 文件载入并执行代码

使用 `source` 函数载入并执行代码，把以下代码放在一个名为 `test.r` 的文件，用文本编辑工具录入。

```
x<-c(22,23,44,66);
y<-mean(x);y
```

然后加载执行，查看输出结果。

```
> source("f:/pro/r/test.r")
> y
[1] 38.75
> x
[1] 22 23 44 66
```

最后将执行结果写入文件。

```
> sink("f:/pro/r/test.lis")
> x
> y
```

打开 test.lis, 可看到以下内容:

```
[1] 22 23 44 66
[1] 38.75
```

如果采用不带参数的 sink, 将恢复结果。示例如下:

```
> sink()
> x
[1] 22 23 44 66
```

#### 4. 代码续行

在行尾使用 “+” 可进行续行。

```
x<-c(11,22,33+
+ 22+
+ 3333)
> x[1] 11 22 3388
```

另外, 需要提一下, R 语言中的注释可以被放在任何地方, 只要是以井号 (#) 开始, 到行末结束就可以。

#### 5. 物件 (对象集)

在 R 中创建的单元为物件 (对象集), 这些物件可以是变量、数字数组、字符串、函数, 以及从这些物件中产生的更多结构。

objects() 可用来显示存储在 R 中的对象集名字。

```
> objects()
[1] "x" "xm"
```

以上代码显示 R 目前运行环境中 x 和 xm 两个变量, 可以使用 rm 移除某个对象。

```
> rm(xm)
> objects()
[1] "x"
```

退出 R 程序时, 可以以 .RData 的方式保存这些对象集, 如图 3-32 所示。

当下次再启动 R 时, 加载 .RData 文件后, 这个对象集会被还原。

```
R version 3.0.0 (2013-04-03) -- "Masked
Marvel"
```



图 3-32 保存对象集对话框



Copyright (C) 2013 The R Foundation for Statistical Computing  
Platform: i386-w64-mingw32/i386 (32-bit)

原来保存的工作空间已还原。

```
> x
[1] 11 22 3388
```

## 3.2.2 向量

### 1. 数据型向量及其运算

最简单的数据结构是数字型向量，它是一个有序的数字集合（这里的序不是指按数字大小排序，是指数字之前的先后顺序都确定），关于这个序，数学上有一个很好的解释，叫作偏序关系。

偏序关系又称半序关系。设  $A$  是一个非空集， $P$  是  $A$  上的一个关系， $P$  适合下列条件：

1) 对任意的  $a \in A, (a, a) \in P$ 。

2) 若  $(a, b) \in P$  且  $(b, a) \in P$ ，则  $a=b$ 。

3) 若  $(a, b) \in P, (b, c) \in P$ ，则  $(a, c) \in P$ ，则称  $P$  是  $A$  上的一个偏序关系。带偏序关系的集合  $A$  称为偏序集或半序集。

比如说 (1,2) 和 (2,1)，它们两个就不是同个向量，因为这两个集合的序不一样。

向量的使用方法很简单，可使用 "c" 后跟括号将向量包围起来，即 c() 函数。如：

```
> y<-c(12,33,12,22)
> y
[1] 12 33 12 22
```

"<-" 可以相当于 "=", 就是将这个向量组作为 y 这个对象的值，也可以使用 assign() 函数。

```
> assign("x",c(11,22,15))
> x
[1] 11 22 15
```

"->" 的作用与 "<-" 类似。

```
> c(12,33,12,22)->z
> z
[1] 12 33 12 22
```

对向量操作，一般是对向量的每个元素进行操作，比如：

```
> z
[1] 12 33 12 22
> z/2
[1] 6.0 16.5 6.0 11.0
```

向量也可以成为 c() 中的参数，向量中的元素，将合并成为 c() 函数中的元素：

```
> c(33,12,66)->x1
> y1=c(x1,111,x1)
> y1
[1] 33 12 66 111 33 12 66
```

再来看看数字型向量运算。向量之间的运算是每个元素分别进行的，比如：

```
> x
[1] 11 22 3388
> 2*x->y
> y
[1] 22 44 6776
> x+3*y->z
> z
[1] 77 154 23716
```

元素个数不一致的向量，元素个数较少的向量将循环扩充和元素个数最多的向量一致，这意味着元素数量最多的向量的元素个数必须是元素数量小的向量的元素个数的整数倍。

```
> z
[1] 77 154 23716
> bb<-c(12,21,32,60,132,56)
> z/3+bb
[1] 37.66667 72.33333 7937.33333 85.66667 183.33333 7961.33333
```

对向量元素的操作，可以使用普通的+、-、\*、/、^等操作符，也可以使用更多的函数，比如：log、sin、tan、max、mean、sum等，这些函数有些是对每个元素分别计算，有些是对所有元素一起计算。

```
> x
[1] 11 22 3388
> cos(x)#cos 三角函数
[1] 0.004425698 -0.999960826 0.206187272
> sin(x)#sin 三角函数
[1] -0.999990207 -0.008851309 0.978512549
> sum(x)# 求和
[1] 3421
> mean(x)# 求平均值
[1] 1140.333
```

可以使用sort、length、sqrt对向量进行排序，求长度，求平方根。

```
> c(4,8,9)->x
> sqrt(x)# 平方根
[1] 2.000000 2.828427 3.000000
> length(x)# 长度
[1] 3
> sort(x)->y# 排序
> y
[1] 4 8 9
```

## 2. 复数向量与规则向量

复数向量的元素都是复数。复数的表示方法是：实部 + 虚部  $i$ 。下面的代码演示了复数向量的使用方法。

```
>c(2+1i,3-9i,4)->b
> y
[1] 4 8 9
> b+y->w# 复数运算
> w
[1] 6+1i 11-9i 13+0i
```

我们可以使用  $1:m-1$  和  $1:(m-1)$  产生规则的序列。

```
>c(1:(22))
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
> c(1:22)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
```

冒号的优先权很高，看下面这个示例，它产生范围在 3~30 之内的公差为 3 的等差数列。

```
> c(3*(1:10))
[1] 3 6 9 12 15 18 21 24 27 30
> c(3*1:10)
[1] 3 6 9 12 15 18 21 24 27 30
```

`seq` 函数是生成序列的最好工具。可以使用它产生符合某种规则的序列，`seq` 函数有 5 个参数，前 4 个参数分别是起始值（参数名称：`from`）、终止值（参数名称：`to`）、步长（参数名称：`by`）、长度即元素个数（参数名称：`length.out`）。

```
> seq(1,5)
[1] 1 2 3 4 5
> seq(1,5,2)
[1] 1 3 5
```

可以直接指定参数名称，传入参数。

```
> seq(from=1,to=15,by=2)
[1] 1 3 5 7 9 11 13 15
>
> seq(from=1,to=15,by=3)
[1] 1 4 7 10 13
> seq(from=1,to=15,length.out=3)
[1] 1 8 15
```

第五个参数是 `along.with`，使用 `along.with` 参数中序列的长度作为要产生序列的长度。

```
> seq(from=2,along.with=c(1:5))
[1] 2 3 4 5 6
> seq(from=2,by=2,along.with=c(1,2,5,8))
[1] 2 4 6 8
> seq(from=2,by=2,along.with=c(1,2,3,8))
[1] 2 4 6 8
```



**注意** along.with 参数中的序列仅取其长度，和序列的内容无关。

rep 函数对序列中的元素进行重复后拼接，拼接的方式是：使用 times 参数将所有元素作为整体拼接，使用 each 参数将元素分别进行拼接。

```
> rep(x,2)
[1] 12 32 98 12 32 98
> rep(x,3)
[1] 12 32 98 12 32 98 12 32 98
> rep(x,times=2)
[1] 12 32 98 12 32 98
> > rep(x,each=2)
[1] 12 12 32 32 98 98
```

### 3. 逻辑型向量

了解了数字型向量，再来看看逻辑型向量。该向量的元素由逻辑型值组成，逻辑型的值有 TRUE（可缩写成 T）、FALSE（可缩写成 F）、NA（即无效）等，可使用 >、>=、==、!= 等逻辑操作符，and 操作作用 &，or 操作作用 |，逻辑非使用 “!”。

```
C(12,33,51)->x
> x
[1] 12 33 51
> x>20->y
> y
[1] FALSE TRUE TRUE
> x>=12->y#x>12 作为产生逻辑向量的依据
> y
[1] TRUE TRUE TRUE
> x>=12&x<30->y
> y
[1] TRUE FALSE FALSE
> x>=12|x<30->y
> y
[1] TRUE TRUE TRUE
>> !(x>=12&x<30)->y
> y
[1] FALSE TRUE TRUE
```

无效值或缺失值 NA、NaN 主要用于应付某操作没完成，结果未知的情况。示例如下：

```
> c(1:4,NA,2:3)->x
> x
[1] 1 2 3 4 NA 2 3
> is.na(x)
[1] FALSE FALSE FALSE FALSE TRUE FALSE FALSE
```

下面是数字计算对 NAN 的处理：

```
> 0/0
[1] NaN
```



```
> 0/0->y
> is.na(y)
[1] TRUE
> y
[1] NaN
```

#### 4. 字符串向量

字符串用单引号或双引号包围，示例如下：

```
> c("qq", "bb")->z
> z
[1] "qq" "bb"
```

可在字符串中使用转义符 \:

```
\n  新行
\r  回车
\t  tab
\b  退格
\a  鸣叫
\\  \
\'  '
\"  "
```

在字符串向量的运算中，paste() 函数接受任意数量的参数，可将它们依次连接到字符串向量的元素中，sep 指定连接时相隔的字符，默认为单个空格。

```
> paste(1:12)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"
> paste("A", 1:6, sep = "")
[1] "A1" "A2" "A3" "A4" "A5" "A6"
> paste("A", 1:6)
[1] "A 1" "A 2" "A 3" "A 4" "A 5" "A 6"
> paste("A", 1:6, sep = " ")
[1] "A1" "A2" "A3" "A4" "A5" "A6"
> paste(c("A", "B"), 1:10, sep="")
[1] "A1" "B2" "A3" "B4" "A5" "B6" "A7" "B8" "A9" "B10"
> paste(" 今天是 ", date())
[1] " 今天是 Sun Apr 21 14:18:38 2013"
```

#### 5. 索引向量

在逻辑值型索引中，索引向量的元素为逻辑值型，逻辑值为 TRUE 的向量将被放在输出结果中。示例如下：

```
> x
[1] 11 22 3388
> x>22->jg
> jg
[1] FALSE FALSE TRUE
```

```
> x[jg]->y
> y
[1] 3388
> x[x<100]->y
> y
[1] 11 22
>
```

在正整数型索引中，索引向量是正整数类型，用于指示要哪些位置的元素要输出到结果中。示例如下：

```
> x[c(1,2,3,2,1)]# 以向量作为索引
[1] 11 22 3388 22 11
> x
[1] 11 22 3388
> x[1]
[1] 11
> x[1:2]
[1] 11 22
> x[2:3]
[1] 22 3388
```

对于负数索引，会将除开索引以外的所有元素输出到结果中。示例如下：

```
> c(12,22,88)->X
> X
[1] 12 22 88
> X
[1] 12 22 88
> X[-(1:2)]->Y
> Y
[1] 88
> X[-(1)]->Y
> Y
[1] 22 88
```

在字符串索引中，是以字符串来标注元素的位置的。示例如下：

```
> c(23,26,27)->age
> c("张三","李四","王五")->names(age)
> age[c("张三")]# 以字符串作为索引
张三
23
> age[c("张三","王五")]->mystudent
> mystudent
张三 王五
23 27
```

带索引方式的向量对象可以直接作为被赋值的对象，所有在索引内的向量元素都会被赋值。示例如下：

```
> age[c("张三","王五")]
张三 王五
```

```

26 28
> age[c("张三","王五")]<-age[c("张三","王五")] +1
> age[c("张三","王五")]
张三 王五
27 29
> age[c("张三","王五")]<-32
> age[c("张三","王五")]
张三 王五
32 32

```

再举个例子：给所有小于 100 的元素均加上 20，代码如下：

```

> x
[1] 11 22 3388
> length(x)
[1] 3
> x[x<100]<-x[x<100]+20
> x
[1] 31 42 3388

```

### 3.2.3 对象集属性

#### 1. 固有属性

对象集的固有属性有 `mode` 和 `length` 两种，相关示例如下：

```

> x
[1] 11 22 3388
> mode(x)
[1] "numeric"

```

其中 `mode` 可理解为对象集的类型，主要有 `numeric1`、`complex`、`logical`、`character` 和 `raw` 等类型。`mode` 的用法如下：

```

> length(x)
[1] 3
> c(1.0-5i,20+51i)->a
> mode(a)
[1] "complex"

```

`mode` 属性转化可完成数据类型的转化。比如，使用 `as.character` 转化为字符型等。示例如下：

```

> h<- 5:12
> h
[1] 5 6 7 8 9 10 11 12
> as.character(h)# 转化为字符
[1] "5" "6" "7" "8" "9" "10" "11" "12"
> c(1.0-5i,20+51i)->a
> mode(a)
[1] "complex"
> as.character(a)->c_a

```

```
> c_a
[1] "1-5i"    "20+51i"
```

## 2. 设置对象属性

可使用 `attr` 方法进行属性的自定义。具体方法为：使用 `attr(object,name)` 格式设置对象属性。示例如下：

```
> h
[1] 5 6 7 8 9 10 11 12
> attr(h,"name")<- "test"
attr(h,"dim")<-c(2,4)
> h
      [,1] [,2] [,3] [,4]
[1,]    5    7    9   11
[2,]    6    8   10   12
attr(,"name")
[1] "test"
```

### 3.2.4 因子和有序因子

因子用来存储类别变量和有序变量，可用来分组或分类，因子表示分类变量，有序因子表示有序变量。

在 R 语言中使用 `factor()` 函数生成因子对象，语法是 `factor(data,levels,labels,...)`，其中 `data` 是数据，`levels` 是因子水平向量，`labels` 是因子的标签向量。

```
> my_num<-c(11,22,34,71,14,68,21)
> factor(my_num)->nums# 生成 my_num 分组向量
```

生成因子对象后，输入对象名称，可显示简单的分类情况。

```
> nums
[1] 11 22 34 71 14 68 21
Levels: 11 14 21 22 34 68 71
>
```

`Levels` 函数用于生成因子向量中的水平（去除重复元素后的元素集）。示例如下：

```
> my_num<-c(11,22,34,71,14,68,21,22,11,34)
> factor(my_num)->nums
> nums
[1] 11 22 34 71 14 68 21 22 11 34
Levels: 11 14 21 22 34 68 71
> levels(nums)
[1] "11" "14" "21" "22" "34" "68" "71"
```

还可使用 `ordered` 函数生成有序因子。示例如下：

```
> ordered(nums)
[1] 11 22 34 71 14 68 21 22 11 34
Levels: 11 < 14 < 21 < 22 < 34 < 68 < 71
> age <- c(25, 12, 15, 12, 25)
```



```
> ordered(age, levels = c(25,12,15))
[1] 25 12 15 12 25
Levels: 25 < 12 < 15
```

cut() 函数将数据转换成因子或有序因子，并进行分组。下面对一组学生成绩进行分组。

```
> score<-c( 88, 85, 75, 97, 92, 77, 74, 70, 63, 97)
> cut(score, breaks = 3)#将成绩分为3组
[1] (85.7,97] (74.3,85.7] (74.3,85.7] (85.7,97] (85.7,97] (74.3,85.7]
[7] (63,74.3] (63,74.3] (63,74.3] (85.7,97]
Levels: (63,74.3] (74.3,85.7] (85.7,97]
> cut(score, breaks = 2)#将成绩分为2组
[1] (80,97] (80,97] (63,80] (80,97] (80,97] (63,80] (63,80] (63,80] (63,80]
[10] (80,97]
Levels: (63,80] (80,97]
```

### 3.2.5 循环语句

#### 1. for 循环

R 语言的 for 循环与 Python 类似，都是通过在对象中进行迭代实现循环，但 R 语言中不能在该循环中直接设置起始值、终止值与步长。示例如下：

```
> z<-c()
> x<-(1:10)
> y<-(11:20)
> for (i in 1:length(x)){
+ z[i]=x[i]^2+y[i]^2
+ }
> z
[1] 122 148 178 212 250 292 338 388 442 500
```

#### 2. while 循环

while 语句每次会检查循环条件，如果条件不再满足，则终止循环。示例如下：

```
> x<-c(1:10)
> i=1
> while (x[i]^2<10)
+ {
+ i=i+1
+ x[i]=x[i]^2
+ }
> x
[1] 1 4 3 4 5 6 7 8 9 10
```

### 3.2.6 条件语句

if...else... 是 R 语言的条件语句。该语句通过检查执行条件来确定是否继续往下执行，如果条件满足，则执行 if 后面的对应语句，否则执行 else 后面的对应语句。示例如下：

```
> z<-c()
```

```

> x<-(1:10)
> y<-(11:20)
> for (i in 1:length(x)){
+ if ((x[i]^3>y[i]^2))
+ z[i]=x[i]^3
+ else
+ z[i]=y[i]^2
+ }
> z
[1] 121 121 144 169 196 225 256 343 512 729 1000

```

## 3.3 R 语言科学计算

### 3.3.1 分类(组)统计

R 语言分类统计主要包括数据分类整理、统计函数定义、数据分类统计 3 个过程。下面以对水果的价格进行分类统计为例说明。

#### 1. 准备分组数据

```

> fruit_class<-c(" 苹果 ", " 梨子 ", " 橘子 ", " 草莓 ", " 苹果 ", " 橘子 ", " 橘子 ", " 草莓 ", " 橘子
", " 草莓 ")
> fruit_prices<-c(3.5,2.5,1.5,5.5,4.2,3.2,2.8,4.8,2.9,5.8)

```

#### 2. 平均价格统计

通过在 `tapply` 函数中指定 `mean` 函数为其参数, 可实现分组求平均值。计算结果分 2 行, 第 1 行为组名, 第 2 行为 `tapply` 函数最后一个函数参数的运算结果。示例如下:

```

> tapply(fruit_prices,fruit_class,mean)
  草莓    橘子    梨子    苹果
5.366667 2.600000 2.500000 3.850000

```

#### 3. 最低价格统计

通过在 `tapply` 函数中指定 `min` 函数为其参数, 可实现分组求最小值。示例如下:

```

> tapply(fruit_prices,fruit_class,min)
  草莓 橘子 梨子 苹果
4.8   1.5  2.5  3.5

```

#### 4. 标准差统计

通过在 `tapply` 函数中指定 `sd` 函数为其参数, 可实现分组求标准差。示例如下:

```

> tapply(fruit_prices,fruit_class,sd)
  草莓    橘子    梨子    苹果
0.5131601 0.7527727      NA 0.4949747

```

#### 5. 标准误

标准误即样本均数的标准差, 是描述均数抽样分布的离散程度及衡量均数抽样误差大小

的尺度,反映的是样本均值之间的变异。

但是,请注意,标准误并不是标准差,而是样本均值的标准差,是用来衡量抽样误差的,其计算公式为:

$$S_x = \frac{S}{\sqrt{n}}$$

其中,  $S$  为样本标准差。标准误越小,表明样本统计量与总体参数的值越接近,样本对总体越有代表性,用样本统计量推断总体参数的可靠程度越大。

可通过在 `tapply` 函数中指定自定义函数 `stderr` 为其参数,来实现分组求标准误。示例如下:

```
> stderr <- function(x) sqrt(var(x)/length(x)) # 自定义 stderr 函数
> tapply(fruit_prices, fruit_class, stderr)
      草莓      橘子      梨子      苹果
0.2962731 0.3763863      NA 0.3500000
```

### 3.3.2 数组与矩阵基础

R 提供了简单的工具以处理数组和矩阵。

#### 1. 数组与矩阵的维数

数组与矩阵的维数是其行向量(或列向量)生成的向量空间的维数,可用维数向量表示,格式为(行数 × 列数),元素都非负。通常使用 `dim` 函数来定义数组维度。示例如下:

```
> dim(my_num) <- c(2, 5) ### 指定数组维度
> my_num
      [,1] [,2] [,3] [,4] [,5]
[1,]   11   34   14   21   11
[2,]   22   71   68   22   34
> dim(my_num) <- c(10)
> my_num
[1] 11 22 34 71 14 68 21 22 11 34
```

#### 2. 切片

切片是操作多维数据(矩阵、数组等)的主要手段,它以索引为参数获取数组或矩阵的一部分。比如:想要得到多维数组的一个切片,则以索引为下标进行访问,取得某块数组。使用 [索引] 格式的参数对数组和矩阵完成切片操作。

索引的形式主要有以下几种:

1) [index1, index2, ..., indexn]。index1、index2 等分别标明了元素在相应维数的索引,将索引组合成完整的位置,标注需要取出的元素,进行切片,形成数组块。比如:数组  $a$  的大小为  $3 \times 5$ ,  $a[2,4]$  表示第 2 行第 4 列的元素。

2) [c(index1, index2, ..., indexn)]。index1、index2 等  $n$  个整数标注了元素的位置,将这些标注的元素取出后,组成数组块。这些元素的位置以列为顺序排列,比如,数组  $a$  的大小为  $3 \times 2$  (3 行 2 列),切片操作  $a[c(1,2,3,4,5,6)]$  依次取出以下元素:  $a[1,1]$ 、 $a[2,1]$ 、 $a[3,1]$ 、 $a[1,2]$ 、 $a[2,2]$ 、 $a[3,2]$ 。

切片操作示例如下：

```
> h
      [,1] [,2] [,3]
[1,]   12   15  982
[2,]   32   67  321
> c(h[1,2],h[2,3])### 获取第 1 行第 2 列数据 (为 15)、第 2 行第 3 列的数据 (为 321)
[1]  15 321
> h[2,]
[1]  32  67 321
> h[c(1,2,3)]### 逐列获取第 1、2、3 个数据
[1] 12 32 15
> h[6]### 逐列获取第 6 个数据
[1] 321
> h[4]
[1] 67
```

### 3. 索引向量

数组可作为索引使用，且数组也是向量，因此作为索引的数组可称为索引向量。下面的代码演示了向量 `c(1:3,5:4,3:5)` 作为索引的情况。

1) 创建数组 `x` 和 `i`，`x` 是被切片的数组，`i` 是索引向量。

```
> array(10:20,dim=c(2,5))->x
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]   10   12   14   16   18
[2,]   11   13   15   17   19
> array(c(1:3,5:4,3:5),dim=c(2,3))->i
> i
      [,1] [,2] [,3]
[1,]    1    3    4
[2,]    2    5    3
```

2) 以 `i` 为索引向量提取数组块，索引向量每个元素代表切片的位置，将这些位置指向的元素提取出来，形成数组块（数组切片）。示例如下：

```
> x[i]
[1] 10 11 12 14 13 12
```

3) 通过索引对数组某个元素赋值。示例如下：

```
> x[i]<-111
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]  111  111  111   16   18
[2,]  111  111   15   17   19
```

### 4. array 函数

`array` 函数根据维数参数生成多维数组，它的参数主要有两个，第一个是需要形成数组元素的数据，第二个是 `dim` 参数提示维度。下面的代码演示了 `array` 函数创建数组的方法，



并通过 `dim` 函数获取数组的大小。

```
> c(1:20)->h
> mya<-array(h,dim=c(4,5))
> mya
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> mydim<-c(2,10)
> mya<-array(h,dim=c(2,10))
> mya
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    3    5    7    9   11   13   15   17   19
[2,]    2    4    6    8   10   12   14   16   18   20
>
> dim(mya)
[1] 2 10
```

`array` 函数的第一个参数既可以是向量也可以是单个值，如下所示：

```
> mya<-array(1,dim=c(2,10))
> mya
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    1    1    1    1
[2,]    1    1    1    1    1    1    1    1    1    1
```

## 5. 数组转换为向量

`as.vector` 函数可将数组转换为向量。示例如下：

```
> x<-array(c(1:10),dim=c(2,5))
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> as.vector(x)
[1] 1 2 3 4 5 6 7 8 9 10
```

## 6. matrix 矩阵

使用 `matrix` 函数可创建矩阵（从数学角度定义的矩阵），主要参数为：`data` 表示构造所需数据，`nrow` 为行数，`ncol` 为列数，`byrow` 表示是否按行顺序分配元素（默认为 `FALSE`，不按）。

```
> matrix(c(1:10),2,5,TRUE)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
> matrix(c(1:10),2,5)
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

## 7. 对角矩阵

对角矩阵是一个除主对角线上的元素之外皆为 0 的矩阵，对角线上的元素可以为 0 或其他值。通过 `diag` 函数可生成和分析对角矩阵，如果参数为一维数组，则将参数视为对角线元素，并生成对角矩阵；如果参数为一维以上数组，则将参数视为对角矩阵，对它进行分析，可提取对角线元素。相关示例如下：

```
> a
[1,] 1 2 3 4 5 6 7 8
>
> diag(a)### 生成对角矩阵
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    0    0    0    0    0    0    0
[2,]    0    2    0    0    0    0    0    0
[3,]    0    0    3    0    0    0    0    0
[4,]    0    0    0    4    0    0    0    0
[5,]    0    0    0    0    5    0    0    0
[6,]    0    0    0    0    0    6    0    0
[7,]    0    0    0    0    0    0    7    0
[8,]    0    0    0    0    0    0    0    8
> a<-array(c(1:16),dim=c(4,4))
> diag(a)### 提取对角线元素
[1,] 1  6 11 16
> a
      [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

## 3.3.3 数组运算

### 1. 四则运算

数组四则运算的规律是：对应位置的元素分别计算，而不是依据矩阵的数学运算法则。运算符为“+”（加）、“-”（减）、“\*”（乘）等，运算优先级与算术四则运算相同，先乘后加减。示例如下：

```
> mya
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    3    5    7    9   11   13   15   17   19
[2,]    2    4    6    8   10   12   14   16   18   20
> myb
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    2    2    2    2    2    2    2    2    2
[2,]    2    2    2    2    2    2    2    2    2    2
> mya+myb
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    3    5    7    9   11   13   15   17   19   21
[2,]    4    6    8   10   12   14   16   18   20   22
> mya*myb
```

```

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    2    6   10   14   18   22   26   30   34   38
[2,]    4    8   12   16   20   24   28   32   36   40
> 3*mya*myb
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    6   18   30   42   54   66   78   90  102  114
[2,]   12   24   36   48   60   72   84   96  108  120
>
> mya*myb+mya# 先乘后加
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    3    9   15   21   27   33   39   45   51   57
[2,]    6   12   18   24   30   36   42   48   54   60
>

```

## 2. 向量连接

向量连接指的是将两个向量通过某种规律连接成一个数组。R 语言的 `cbind` 和 `rbind` 函数可进行向量连接，其中 `cbind` 函数将向量的行转变为列后再连接，`rbind` 函数将向量的列转变为行后再连接。示例如下：

```

> x2<-c(101:105)
> x1<-c(1:10)
> cbind(x1,x2)
      x1 x2
[1,]  1 101
[2,]  2 102
[3,]  3 103
[4,]  4 104
[5,]  5 105
[6,]  6 101
[7,]  7 102
[8,]  8 103
[9,]  9 104
[10,] 10 105
> rbind(x1,x2)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
x1    1    2    3    4    5    6    7    8    9   10
x2   101   102   103   104   105   101   102   103   104   105

```

## 3.3.4 矩阵运算

### 1. 矩阵连接

R 语言的 `cbind` 函数完成矩阵的横向连接，`rbind` 函数完成矩阵的纵向连接。下面是关于矩阵的连接操作示例。

```

> x3<-matrix(c(1:10),2,5)
> x4<-matrix(c(101:105),2,5)
> x3
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9

```

```

[2,] 2 4 6 8 10
> x4
      [,1] [,2] [,3] [,4] [,5]
[1,] 101 103 105 102 104
[2,] 102 104 101 103 105
> cbind(x3,x4)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 1 3 5 7 9 101 103 105 102 104
[2,] 2 4 6 8 10 102 104 101 103 105
> rbind(x3,x4)
      [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
[3,] 101 103 105 102 104
[4,] 102 104 101 103 105

```

## 2. 矩阵转置

线性代数将矩阵  $A$  的转置 (记做  $A^T$ ) 定义为:

把  $A$  的横行写为  $A^T$  的纵列;

把  $A$  的纵列写为  $A^T$  的横行。

根据上述计算法则,  $m \times n$  矩阵  $A$  的转置生成  $n \times m$  矩阵  $A^T$ 。

$$A_{ij}^T = A_{ji}, 1 \leq i \leq n, 1 \leq j \leq m。$$

R 语言的 `t` 函数可完成矩阵转置计算。

```

> array(h,dim=c(2,5))->mya
> mya
      [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
> t(mya)### 矩阵转置
      [,1] [,2]
[1,] 1 2
[2,] 3 4
[3,] 5 6
[4,] 7 8
[5,] 9 10

```

相对 `t` 函数而言, 用 `aperm` 函数进行矩阵转置更灵活。`aperm` 有两个常用的参数, 第一个参数是需要转置的矩阵, 第二个参数 `perm` 指示新矩阵相对于第一个参数矩阵的维度下标。需要特别注意的是, 第二个参数 `perm` 是维度下标。比如, 将行转换为列, 将列转换为行, 将行列次序更换, 将第一维的元素与第二维的元素互换, 则将 `perm` 设为 `c(2,1)`。下面的代码演示了 `aperm` 函数的使用方法。

```

> array(h,dim=c(2,5))->mya
> mya
      [,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
>
> aperm(mya,perm=c(2,1))->myb### 以 c(2,1) 为维度下标进行转置

```



```
> myb
```

```
      [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
```

```
>
```

```
> array(mya,c(2,2,5))->mya1
```

```
> mya1
```

```
.,, 1
```

```
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
.,, 2
```

```
      [,1] [,2]
```

```
[1,]    5    7
[2,]    6    8
```

```
.,, 3
```

```
      [,1] [,2]
```

```
[1,]    9    1
[2,]   10    2
```

```
.,, 4
```

```
      [,1] [,2]
```

```
[1,]    3    5
[2,]    4    6
```

```
.,, 5
```

```
      [,1] [,2]
```

```
[1,]    7    9
[2,]    8   10
```

```
> aperm(mya1,perm=c(2,1,3))->myb1### 以 c(2,1,3) 为维度下标进行转置
```

```
> myb1
```

```
.,, 1
```

```
      [,1] [,2]
```

```
[1,]    1    2
[2,]    3    4
```

```
.,, 2
```

```
      [,1] [,2]
```

```
[1,]    5    6
[2,]    7    8
```

```
.,, 3
```

```

[1,] [1,2]
[1,] 9 10
[2,] 1 2
, , 4
[1,] [1,2]
[1,] 3 4
[2,] 5 6
, , 5
[1,] [1,2]
[1,] 7 8
[2,] 9 10
> aperm(mya1,perm=c(1,3,2))->myb1

```

```
> myb1
```

```
, , 1 矩阵即矩阵A的转置(记做 $A'$ )定义为:
```

把A的横行写为 $A'$ 的纵列:

```

[1,] [1,2] [1,3] [1,4] [1,5]
[1,] 1 5 9 3 7
[2,] 2 6 10 4 8

```

R语言的t函数可完成矩阵转置计算。

```
, , 2
```

```

[1,] [1,2] [1,3] [1,4] [1,5]
[1,] 3 7 1 5 9
[2,] 4 8 2 6 10
>

```

### 3. 矩阵乘积

若 $A$ 为 $m \times n$ 矩阵, $B$ 为 $n \times r$ 矩阵,则它们的乘积 $AB$ (有时记做 $A \cdot B$ )会是一个 $m \times r$ 的矩阵,前提是 $m$ 与 $n$ 必须相同,矩阵乘积使用`%*`操作符进行计算。示例如下:

```

> a
[1,] [1,2] [1,3] [1,4] [1,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
> b
[1,] [1,2]
[1,] 1 6
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10
> a %*% b
[1,] [1,2]
[1,] 95 220
[2,] 110 260

```

#### 4. 内外积运算

1) 向量外积。向量的外积是矩阵的克罗内克积的特殊情况。给定  $m \times 1$  列向量  $u$  和  $1 \times n$  行向量  $v$ , 它们的外积  $u \otimes v$  被定义为  $m \times n$  矩阵  $A$ 。

$$u \otimes v = A = uv$$

向量外积  $u \otimes v$  的计算定义为:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \otimes \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_2 b_1 & a_3 b_1 \\ a_1 b_2 & a_2 b_2 & a_3 b_2 \\ a_1 b_3 & a_2 b_3 & a_3 b_3 \\ a_1 b_4 & a_2 b_4 & a_3 b_4 \end{bmatrix}$$

下面的代码演示了  $a$  和  $b$  数组作为向量的外积运算和普通乘法运算。

```
> b<-array(c(1:4))
```

```
> a<-array(c(5:6))
```

```
> b%o%a### 外积
```

```
 [,1] [,2]
```

```
[1,] 5 6
```

```
[2,] 10 12
```

```
[3,] 15 18
```

```
[4,] 20 24
```

```
> b
```

```
[1] 1 2 3 4
```

```
> a
```

```
[1] 5 6
```

```
> b<-array(c(1:4))
```

```
> a<-array(c(5:8))
```

```
> a*b### 普通乘法
```

```
[1] 5 12 21 32
```

```
> b
```

```
[1] 1 2 3 4
```

```
> a
```

```
[1] 5 6 7 8
```

```
> a%o%b
```

```
 [,1] [,2] [,3] [,4]
```

```
[1,] 5 10 15 20
```

```
[2,] 6 12 18 24
```

```
[3,] 7 14 21 28
```

```
[4,] 8 16 24 32
```

```
>
```

此外, 还可以使用 `Outer(a,b, “*”)` 替代 `%o%` 运算符进行外积运算。

2) 向量内积。向量内积以实数  $\mathbb{R}$  上定义的两个向量为运算对象, 返回一个实数标量值, 属于二元运算, 它是欧几里得空间的标准内积。

两个向量  $a = [a_1, a_2, \dots, a_n]$  和  $b = [b_1, b_2, \dots, b_n]$  的内积定义为:

$$a \cdot b = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

R 语言通过 `crossprod` 函数完成向量内积计算。示例如下：

```
> a<-c(1:3)
> b<-c(4:6)
> crossprod(a,b)
      [,1]
[1,]    32
> a<-c(1:3)
```

3) 矩阵内积。矩阵内积的计算方式相当于第一个参数的转置乘以第二个参数，就是前面提到的矩阵乘积。除了使用 `%%` 操作符外，还可以使用 `crossprod` 函数完成矩阵内积计算。示例如下：

```
> b<-array(c(4:6),dim=c(1,3))
> a<-array(c(1:3),dim=c(1,3))
> a
      [,1] [,2] [,3]
[1,]    1    2    3
> b
      [,1] [,2] [,3]
[1,]    4    5    6
> crossprod(a,b)####a 与 b 完成矩阵内积计算
      [,1] [,2] [,3]
[1,]    4    5    6
[2,]    8   10   12
[3,]   12   15   18
> t(a) %*% b####a 的转置与 b 完成矩阵内积计算
      [,1] [,2] [,3]
[1,]    4    5    6
[2,]    8   10   12
[3,]   12   15   18
```

## 5. 求解线性方程组

一般通过 `solve` 函数来求解  $a \%*\% x = b$  中的  $x$  向量值，求解线性方程组仅使用 `solve` 函数的两个参数，第一个  $a$  为系数矩阵，第二个  $b$  为常数项，当  $b$  缺失时，默认为单位矩阵。示例如下：

```
> b
      [,1]
[1,]    8
[2,]    9
> a
      [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(a,b)
      [,1]
[1,] -2.5
[2,]  3.5
```

## 6. 矩阵求逆

通过 `solve` 函数可进行矩阵求逆计算，只指定 1 个参数（待求逆的矩阵）即可。示例如下：

```
> a
      [,1] [,2]
[1,]    1    3
[2,]    2    4

> solve(a)
      [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
>
```

## 7. 矩阵的特征值求解

1) 特征值概念。 $\lambda$  是  $A$  的特征值等价于线性系统  $(A - \lambda I) v = 0$  (其中  $I$  是单位矩阵) 有非零解  $v$  (一个特征向量), 特征值存在等价于下面行列式成立:

$$\det(A - \lambda I) = 0$$

函数  $p_A(\lambda) = \det(A - \lambda I)$  是一个关于  $\lambda$  的多项式, 称为  $A$  的特征多项式。矩阵的特征值也就是其特征多项式的零点。

求一个矩阵  $A$  的特征值可以通过求解方程  $p_A(\lambda) = 0$  来得到。

2) R 语言求解特征值。利用 R 语言的 `eigen` 函数可求解特征值, 调用格式如下:

```
eigen(x, symmetric, only.values = FALSE)
```

其中,  $x$  为要求特征值的矩阵; `symmetric` 是逻辑型, 表示是否为对称矩阵, 对称矩阵是一个方形矩阵, 其转置矩阵和自身相等, 即:  $A = A^T$ , 对称矩阵  $A = (a_{ij})$  从右上至左下方向的元素以主对角线 (左上至右下) 为轴对称, 即:  $a_{ij} = a_{ji}$ 。 `only.values` 如果为 `TRUE`, 则只返回特征值, 否则返回特征值和特征向量。

下面的代码演示了 `eigen` 函数计算特征值的方法。

```
> a<-array(c(1:16),dim=c(4,4))
> eigen(a)### 计算 a 的特征值
$values
[1]  3.620937e+01 -2.209373e+00  1.599839e-15  7.166935e-16

$vectors
      [,1]      [,2]      [,3]      [,4]
[1,] 0.4140028  0.82289268 -0.5477226  0.1125155
[2,] 0.4688206  0.42193991  0.7302967  0.2495210
[3,] 0.5236384  0.02098714  0.1825742 -0.8365883
[4,] 0.5784562 -0.37996563 -0.3651484  0.4745519

>
> eigen(a,only.values=FALSE)
$values
[1]  5.3722813 -0.3722813

$vectors
      [,1]      [,2]
```



```
[1,] -0.5657675 -0.9093767
[2,] -0.8245648  0.4159736
```

## 8. 求解矩阵行列式

行列式是线性代数中的一个概念，将一个  $n \times n$  的矩阵  $A$  映射到一个标量，记作  $\det(A)$  或  $|A|$ 。行列式可看作是有向面积的概念在一般的欧几里得空间中的推广。

比如，假设矩阵  $A$  定义为：

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

则矩阵  $A$  的行列式  $|A|$  可定义为：

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

R 语言的  $\det$  函数可求解矩阵对应的行列式值，即：已知矩阵  $A$ ，求解  $|A|$ 。示例如下：

```
> a<-array(c(1:4),dim=c(2,2))
> det(x)
[1] -2
>
```

## 9. 奇异分解

奇异值分解是线性代数中一种重要的矩阵分解，在信号处理、统计学等领域有重要应用。假设  $M$  是一个  $m \times n$  阶矩阵，其中的元素全部属于域  $K$ （实数域或复数域）。设存在一个分解使得：

$$M = U \Sigma V^*$$

其中  $U$  是  $m \times m$  阶酉矩阵； $\Sigma$  是半正定  $m \times n$  阶对角矩阵； $V^*$ （即  $V$  的共轭转置）是  $n \times n$  阶酉矩阵。

这种分解称作  $M$  的奇异值分解， $\Sigma$  对角线上的元素  $\Sigma_{i,i}$  即为  $M$  的奇异值。

使用 R 语言的  $\text{svd}$  函数可完成奇异分解。示例如下：

```
> array(c(1:16),dim=c(4,4))>a
> a
      [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> svd(a)### 奇异分解
$d
[1] 3.862266e+01 2.071323e+00 1.291897e-15 6.318048e-16
$u
      [,1]      [,2]      [,3]      [,4]
[1,] -0.4284124 -0.7186535  0.5462756 -0.0397869
```

```
[2,] -0.4743725 -0.2738078 -0.6987120 0.4602190
[3,] -0.5203326 0.1710379 -0.2414027 -0.8010772
[4,] -0.5662928 0.6158835 0.3938391 0.3806452
$V
      [,1]      [,2]      [,3]      [,4]
[1,] -0.1347221 0.82574206 -0.4654637 -0.2886928
[2,] -0.3407577 0.42881720 0.4054394 0.7318599
[3,] -0.5467933 0.03189234 0.5855124 -0.5976414
[4,] -0.7528288 -0.36503251 -0.5254881 0.1544743
```

## 3.4 R 语言计算实例

### 3.4.1 学生数据集读写

下面演示 R 语言对学生数据集的操作。R 语言可以使用 list (列表) 组件创建与读写学生数据, 该组件通常用来容纳一个数据集, 其中包含不同的数据类型。

1) 创建学生数据集 (创建列表的语法是: list (字段 1= 组件 1, 字段 2 = 组件 2, ...)), 学生数据集由 3 个不同类型的数据组成: name (姓名, 类型为字符型)、class (班级, 类型为字符型)、ages (年龄, 类型为数值型)。

```
> list(name="students",class="101",stdt.ages=c(22,25,20),stdt.name=c("zhangsang",
"lisi","wangwu"))->mystudents
```

2) 读取列表。下面的代码读取刚才创建的学生数据集:

```
> mystudents
$name
[1] "students"
```

```
$class
[1] "101"
```

```
$stdt.ages
[1] 22 25 20
```

```
$stdt.name
[1] "zhangsang" "lisi"      "wangwu"
```

3) 获取学生数据集的字段总数 (通过 length 返回 list 组件的数量)。

```
> length(mystudents)
[1] 4
```

4) 查看数据集中所有学生的姓名和年龄 (通过 “列表变量名 \$ 字段名” 提取组件内容)。

```
> c(mystudents$stdt.name,mystudents$stdt.ages)
[1] "zhangsang" "lisi"      "wangwu"    "22"        "25"        "20"
```

此外, R 语言还提供了一个很不错的 list 组件 data.frame, 它内部可拥有很多组件。下面接着以学生数据为例讲解 data.frame。

## 1) 创建 data.frame 组件，存储学生数据。

```
>
data.frame(name=mystudents$stdt.name,age=mystudents$stdt.ages)->mysts
> mysts
  name age
1 zhangsang 22
2 lisi 25
3 wangwu 20
```

2) 将数据集中的年龄都增长 1 岁（随着新的一年到来，学生们都长大了 1 岁），完成这个操作可使用 attach 和 detach 方法。

前面一直用 \$ 符号访问 list 列表的字段，当 R 语言的代码较多时，列表组件名前缀访问字段很不方便，因此，R 语言提供了另一对非常有用的工具 attach 和 detach：attach 把数据集的所有字段复制一份副本，绑定在搜索路径，这样可以直接读取它们（仅能读取，写回没有意义，因为这只是副本而已），无需显示表明列表名字；detach 则进行解绑。

## 1) 用 attach 将学生数据集的字段副本绑定在搜索路径中。

```
> attach(mysts)
> age
[1] 22 25 20
> name
[1] zhangsang lisi wangwu
Levels: lisi wangwu zhangsang
```

## 2) 将绑定的 age 字段副本加 1，并显示更新后的学生数据。

```
> age+1->mysts$age
> mysts
  name age
1 zhangsang 23
2 lisi 26
3 wangwu 21
```

## 3) 使用 detach 将字段副本从搜索路径上删除（解绑）。

```
> detach(mysts)
> age
错误：找不到对象 'age'
> name
错误：找不到对象 'name'
```

## 3.4.2 最小二乘法拟合

## 1. 最小二乘法与回归

最小二乘法是一种数学优化技术，它通过最小化误差的平方和找到一组数据的最佳函数匹配。

假设存在  $(x, y)$  这两个变量，对于一系列的  $x$  变量值，有一系列的  $y$  值与其对应，可以找到这两个变量之间的相互关系。比如：对于一次函数来说，可将这些  $(x, y)$  值标注在直角

坐标系统，从而得到一条直线，这些点就在这条直线附近。那么，直线方程的定义为：

$$y=kx+b$$

其中： $k$ 、 $b$  是任意实数， $k$  为斜率， $b$  为截距。

下面以  $y_1=3x+12$  和  $y_2=6x+12$  为例进行分析。如图 3-33 所示，实线为  $y_1$  的图像，虚线为  $y_2$  的图像。从图像能直观看出，斜率越大，直线越陡。 $y_1$  的斜率是 3，截距为 12； $y_2$  的斜率是 6，截距为 12， $y_2$  方程的图像明显比  $y_1$  陡。

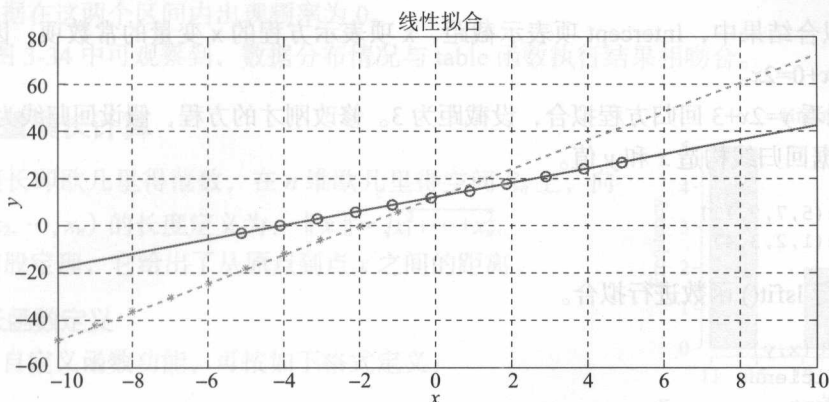


图 3-33 直线的图像

图 3-33 中标注的圆圈和星号为  $(x, y)$  格式表示的二维数据点，很明显，圆圈的数据点位于  $y_1=3x+12$  上，而星号的数据点位于  $y_2=6x+12$  上。这些二维数据点被  $y_1=3x+12$  和  $y_2=6x+12$  方程的图像连接。这样做的好处是，我们不需要记忆这些数据点的坐标就能预测类似数据点的位置。比如说，已知  $x$  为 1.5 时，想要求圆圈数据点的坐标，可直接将  $x=1.5$  代入  $y_1$  方程得到：

$$y=1.5 \times 3+12=16.5$$

这样就可用  $y=kx+b$  形式的一次方程拟合数据点，这个过程为线性拟合。拟合的目标是这些点到这条直线的距离的平方和最小。最小二乘法是效果较好的线性拟合方法，最小二乘法拟合数据点的过程就是对数据做回归分析，我们把类似图中的这几条直线称为回归线。

## 2. 最小二乘法拟合

R 语言提供了 `lsfit` 函数，可完成最小二乘法拟合，其主要参数如下。

□ **X**：一个矩阵的行对应的情况和其列对应为变量。

□ **Y**：结果，可以是一个矩阵。

□ **Wt**：可选参数，加权最小二乘法的执行权重向量。

□ **Intercept**：是否应使用截距项。

□ **Tolerance**：公差将用于矩阵分解。

□ **Yname**：用于响应变量的名称。

下面来看看  $y=2x$  回归方程拟合，这里以  $x=(1,2,3,4)$ ， $y=(2,4,6,8)$  为例在 R 中进行

数据拟合。

```
> y<-c(2,4,6,8)
> x<-c(1,2,3,4)
> lsfit(x,y)### 下面的 x 为常数项, Intercept 为截距
$coefficients
Intercept      x
          0          2
.....
```

上述拟合结果中, Intercept 项表示截距, x 项表示方程的 x 变量的常数项, 因此, 回归方程为  $y=2x+0=2x$ 。

再来看看  $y=2x+3$  回归方程拟合, 设截距为 3。修改刚才的方程, 假设回归线为:  $y=2x+3$ 。

1) 根据回归线构造 x 和 y 值。

```
> y<-c(5,7,9,11)
> x<-c(1,2,3,4)
```

2) 执行 lsfit() 函数进行拟合。

```
> lsfit(x,y)
$coefficients
Intercept      x
          3          2
```

上面 lsfit() 函数的运行结果表明, 这些数据点的回归方程为  $y=2x+3$ 。

### 3.4.3 交叉因子频率分析

交叉因子频率分析的作用在于分析数据的分布区间及其统计指标。下面举例说明分析过程。

1) 划分数据分布区间。使用 cut 函数将变量 y 中存储的数字划分到 5 个分布区间: [11,15]、[15,19]、[19,23]、[23,27]、[27,31] 示例如下:

```
> y<-c(11,22,13,14,11,22,31,31,31,14)
> cut(y,5)->cuty
> cuty
[1] (11,15] (19,23] (11,15] (11,15] (11,15] (19,23] (27,31] (27,31] (27,31]
[10] (11,15]
Levels: (11,15] (15,19] (19,23] (23,27] (27,31]
```

2) 使用 table 函数统计数据在每个区间出现的频率。代码如下:

```
> table(cuty)
cuty
(11,15] (15,19] (19,23] (23,27] (27,31]
      5         0         2         0         3
```

3) 使用 hist 函数生成分布直方图, 以便更直观地观察数据分布情况, 如图 3-34 所示。通过指定 breaks 参数 (设置为各区间的边界值) 和 axes 参数 (设置为 FALSE 表示手动画刻度), 将数据在 table 函数生成的区间内进行划分。代码如下:



```
> bins<-seq(min(y),max(y),by=4)
> hist(y,breaks=bins,col="lightblue",axes=FALSE)
> axis(1,bins)
> axis(2)
```

结合 table 函数的执行结果以及 hist 函数生成的直方图,可得到以下结论:

1) 分析 table 函数的执行结果可看出,数据主要集中在 [11,15] 区间中,[11,15] 区间内分布的数字最多,该区间内有 5 个数字。此外,在 [15,19]、[23,27] 区间中没有数据分布,变量  $y$  中的数据在这两个区间内出现频率为 0。

2) 从图 3-34 中可观察到,数据分布情况与 table 函数执行结果相吻合。

### 3.4.4 向量模长计算

向量模长即欧几里得范数,在  $n$  维欧几里得空间  $R_n$  上,向量  $x = (x_1, x_2, \dots, x_n)$  的长度定义为:  $\|x\| = \sqrt{x_1^2 + \dots + x_n^2}$ 。

根据勾股定理,它给出了从原点到点  $x$  之间的距离。

#### 1. 模长函数定义

R 拥有自定义函数功能,可按如下格式定义:

```
函数名 <-function(参数1,参数2,...,参数n){
  函数体
}
```

下面定义求三维向量模长的 vector\_length 函数,完成模长计算。

```
> vector_length<-function(x1,x2,x3){
+ vlength<-sqrt (x1^2+x2^2+x3^2)
+ vlength
+ }
```

#### 2. 模长计算

调用 vector\_length 函数,计算向量 [12,33,19] 的模长。

```
> vector_length(12,33,19)
[1] 39.92493
>
```

$N$  维向量的模长计算与三维模长类似。下面重新定义 vectorn\_length 函数,并调用它计算任意维度向量的模长。

```
> vectorn_length<-function(x){
+ temp<-0
+ for (i in 1:length(x)){
+ temp<-temp+x[i]^2
+ }
+ vlength<-sqrt(temp)
+ vlength
+ }
```

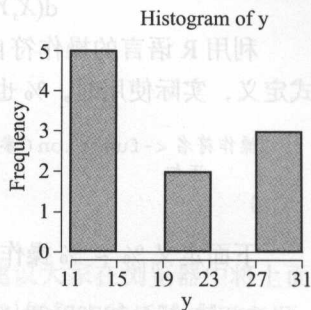


图 3-34 分布直方图

```
> # 下面调用新的 vectorn_length 函数计算模长
> vectorn_length(c(11,22,33,44,55))
[1] 81.57818
> vectorn_length(c(11,22,33,55))
[1] 68.69498
```

### 3.4.5 欧氏距离计算

欧氏距离 (Euclid Distance) 是在  $n$  维空间中两个点之间的真实距离,  $n$  维欧氏空间的每个点可以表示为  $(x[1], x[2], \dots, x[n])$ ,  $X = (x[1], x[2], \dots, x[n])$  和  $Y = (y[1], y[2], \dots, y[n])$  这两个点之间的距离  $d(X, Y)$  定义为下面的公式:

$$d(X, Y) = \sqrt{\sum ((x[i] - y[i])^2)} \text{ 其中 } i = 1, 2, \dots, n$$

利用 R 语言的操作符自定义功能完成欧氏距离计算, 操作符的定义使用 % 符号 % 的方式定义, 实际使用时, % 也属于操作符的一部分。操作符的定义格式如下:

```
操作符名 <- function(参数 1, 参数 2, ..., 参数 n){
  语句
}
```

下面定义 % ~ % 操作符, 并计算二维空间的欧氏距离。示例如下:

```
> "%~%" <- function(x1, x2){
+ temp <- 0
+ for (i in 1:length(x1)){
+ temp <- temp + (x1[i] - x2[i])^2
+ }
+ edis <- sqrt(temp)
+ edis
+ }
> # 使用 %~% 操作符, 计算二维空间的欧氏距离
> c(1,2,3) %~% c(5,6,7)
[1] 6.928203
>
```

可以将计算扩展到  $n$  维空间中。使用 R 语言的不定数量的函数参数机制, 来定义  $n$  维空间的欧氏距离函数 mycount。示例如下:

```
> mycount <- function(...){
+ temp = 0
+ for (i in c(...)){
+ temp = temp + 1
+ }
+ temp
+ }
> mycount(11,22,33)
[1] 3
> mycount(11,22,33,66)
[1] 4
> mycount(11,22,66)
[1] 3
```

### 3.5 小结

本章对 Python 语言和 R 语言的语法基础以及相关计算平台 API 进行了讲述，同时，用大量实例讲解了相关计算平台的实际操作。

Python 语言和 R 语言是本书讲解机器学习用到的主要语言，也是机器学习工程应用中可能用到的编程语言，在此建议大家平时多阅读 R 语言和 Python 语言的官网教程和相关计算平台资料，加深对它们的理解。

因本书篇幅有限，更多的关于 R 语言和 Python 语言的资料可以查询相关官网。下面列举了常用的官网链接。

R 语言文档资料链接：

<http://cran.r-project.org/manuals.html>

Python 科学计算库文档资料链接：

<http://docs.python.org/2/>

<http://docs.scipy.org/doc/>

<http://docs.opencv.org/master/modules/refman.html>

从下章开始，我们将正式进入机器学习和统计分析的实战。建议大家在浏览器中将上面的文档链接收藏，以便更好地理解本书内容。此外，从本章开始，每章小结后均有思考题，希望大家在阅读本书的过程中，多动手，多上机操作，理论联系实践才是王道。

### 思考题

(1) 本章中提到了 Python 将信息隐藏在声音和图像载体文件的方法，能不能将隐藏了信息的图像载体文件隐藏在一段音乐之中？



提示 可以考虑先将隐藏了信息的图像矩阵读入，然后将其作为原始数据混进 WAV 格式的音乐文件之中，再用原始音乐文件作为解码的密钥，解码图像数据后，用本章介绍的方法从图像数据中恢复文字信息。

(2) 用 R 语言分析一个数据集的数据，将数据分为适当的区间，然后统计数据在每个区间的分布数量，并作出直方图。



提示 用因子频率分析方法实现。

(3) 用 R 语言在  $x$  和  $y$  之间建立回归模型，得出回归直线方程， $x=[1,3,8,9]$ ,  $y=[2,8,23,80]$



提示 使用 R 语言的回归计算函数分析。

## 生产环境基础

机器学习的任务是研究计算机怎样模拟或实现人类的学习行为，重新组织已有的知识结构使之不断改善自身的性能，它是人工智能的核心，是使计算机具有智能的根本途径。

从目前的发展情形来看，机器学习主要依靠各种算法来实现，比如：神经网络、SVM、决策树、K近邻、K-Means、回归算法等。而算法是对特定问题求解过程的描述，是为解决某一特定问题而采取的具体有限的操作步骤。随着计算机科学技术的发展，算法在计算机方面已有了广泛的发展及应用。算法可理解为计算机指令的有限序列，每条指令完成一个或多个操作，它是描述计算机程序行为的语言，是让程序变得最为简洁的思考方式。程序在调试完毕后，需要投入到实际运行阶段，这个阶段需要一个运行平台，这就是生产环境。

在工业界，生产环境主要指生产现场中进行制造的地点，包括生产工装、量具、工艺过程、材料、操作者、环境和过程设置。在软件工程领域界，“生产环境”一词是指软件调试完毕并正式启用后实际运行的环境。在该环境下，软件系统运行的目标是稳定、安全、可靠。目前，生产环境中最常用的是 Windows Server 和 Linux/UNIX 两种，考虑到读者普遍比较熟悉 Windows 系统，本章将重点讲解 Linux 下的生产环境基础。

### 4.1 Windows Server 2008 基础

以 Windows Server 2008 作为生产环境，可开发、提供和管理丰富的企业级应用程序，充分利用其提供的高度安全的网络基础架构，提高和增加技术的效率与价值。Windows Server 2008 建立在网络和虚拟化技术之上，可提高基础服务器设备的可靠性和灵活性。新的虚拟化工具、网络资源和增强的安全性等均可降低成本，为动态和优化的数据中心提供平台。故障转移集群的改进可简化集群，提高集群的稳定性并使得它们更加安全，新的故障转



移集群验证向导可用于测试存储。此外, Windows Server 2008 还包括一个新的 TCP/IP 协议栈, 称为下一代 TCP/IP 协议栈。下一代 TCP/IP 协议栈完全重新设计了 TCP/IP 功能, 兼容了互联网协议第 4 版 (IPv4) 和互联网协议第 6 版 (IPv6), 符合不同的网络环境和技术的连通性和性能需要。

#### 4.1.1 Windows Server 2008 R2 概述

Windows Server 2008 是专为强化下一代网络、应用程序和 Web 服务的功能而设计的, 是有史以来最先进的 Windows Server 操作系统。与 Windows Server 2008 相比, Windows Server 2008 R2 继续拓展了虚拟化、系统管理弹性、网络存取方式, 以及信息安全等领域的应用。Windows Server 2008 R2 中重要的新功能包含: Hyper-V 加入了动态迁移的功能, 作为最初发布版中快速迁移功能的一个改进, Hyper-V 将以毫秒计算迁移时间, 与 VMware 公司的 ESX 或其他管理程序相比, 它是 Hyper-V 功能的一个强项。此外, 它还强化了 PowerShell 对各个服务器角色的管理指令。该系统具有以下特色:

- ❑ Hyper-V 2.0: 使虚拟化的功能与可用性更完备。Hyper-V 2.0 不仅支持 Live Migration 动态迁移, 还能支持更多的 Linux 操作系统安装在 VM 上。Windows Server 2008 推出半年后, 微软就推出了内建在 Windows Server 2008 上的虚拟化平台 Hyper-V 1.0, 这个版本虽然具有基本的虚拟化功能, 但是相比于其他虚拟化平台却薄弱许多, 如缺乏动态迁移功能, 因此无法在不停止虚拟主机 (VM) 的情况下, 将 VM 转移到其他实体服务器上。目前, 这项功能在 Windows Server 2008 R2 的 Hyper-V 2.0 上开始支持, 使得这项虚拟化平台的可用性迈进了一大步。
- ❑ Active Directory Administrative Center: 可离线加入网域、AD 资源回收筒 (AD 可强化管理接口与部署弹性)。Active Directory (AD) 在 Windows Server 操作系统中从来都是举足轻重的服务器角色, Windows Server 2008 R2 对此也强化了不少功能。例如具有新的 AD 管理接口, 同时还能使用 PowerShell 指令操作; 可让计算机离线加入网域, 并有 AD 资源回收站, 增加了 AD 成员的增删弹性。
- ❑ Windows PowerShell 2.0 与 Server Core: Server Core 模式支持 .NET, R2 改善了 Server Core 不支持 .NET Framework 且不能使用 PowerShell 的缺点。现在在指令操作为主要要求的 Server Core 中, 可以搭配 PowerShell, 从而使服务器管理的操作更有效率。Server Core 安装选项是安装 Windows Server 2008 操作系统的一种新的选择。Server Core 安装提供了一个最小的运行环境作为特定的服务器角色, 降低了服务器的维护管理强度, 同时降低了服务器角色被黑客攻击的可能性。
- ❑ Remote Desktop Services: 可提升桌面与应用程序的虚拟化功能。在新版的 RDS 中, 也增加了新的 Remote Desktop Connection Broker (RDCB)。这项功能可整合 RDS 所有的应用程序服务器, 包含实体主机和 VM。
- ❑ DirectAccess: 可提供更方便、更安全的远程联机通道。DirectAccess 让 VPN 通道的建立变得更加简便, 可整合多种验证机制及 NAP, 有助于提高联机过程中的安全性。



❑ BranchCache：可加快分公司之间档案存取的新做法。利用档案快取的方式，可以就近存取先前已经下载过的档案，除了能更快地取得分享数据之外，也能减少对外联机频宽的浪费。

❑ URL-based QoS：企业可进一步控管网页存取频宽。企业可以针对所有个人计算机连接特定网站的联机定义优先权，加快重要网页的存取速度。

❑ BitLocker to Go：支持可移除式存储装置加密。BitLocker to Go 加密随身碟这一步骤的特别之处在于可以整合智能卡验证使用者身份的真实性，使得存储装置的控管变得更加安全。

❑ AppLocker：可提高个人端应用程序的控管度。AppLocker 可称为软件限制原则的加强版本，除了具备一切旧有功能，最为重要的是企业可以通过不可随意修改的发行者信息，有效地禁止或允许应用程序的执行，同时也更加完善了其自身的安全性能。

#### 4.1.2 Windows PowerShell

Windows PowerShell 是微软公司为 Windows 环境所开发的壳程序 (Shell) 及脚本语言技术，采用的是命令行界面。这项全新的技术提供了丰富的控制选项与自动化的系统管理能力。Microsoft 推出 Windows PowerShell 的目的是：使 Windows PowerShell 成为相当于 UNIX/Linux 系统的命令行壳程序 (如 sh、bash 或 csh)，同时还可内置脚本语言及辅助脚本程序的工具。

Windows PowerShell 以 .NET Framework 技术为基础，并且与现有的 WSH 保持向后兼容，因此它的脚本程序不仅能访问 .NET CLR，也能使用现有的 COM 技术。同时还包含了数种系统管理工具，简易且一致的语法，可提升管理者的处理能力，常见的如登录数据库、WMI 等。Windows PowerShell 具有以下优势：

- ❑ 一致性的设计让所有的工具和系统数据的使用语法、命名原则都相同。
- ❑ 脚本语言简单易学，而且还能支持现有的脚本程序和命令行工具。
- ❑ 内含 129 种被称为 cmdlet 的标准工具，可用来处理常见的系统管理工作。
- ❑ 具备完整的可扩展性，独立软件商或开发者都能很容易地根据需求自行扩充。
- ❑ 进程间数据传递的内容具有强类型特征。

以下示例演示了 Windows PowerShell 的基本使用方法。

获取所有命令：

```
PS> Get-Command
```

查看 Get-Command 命令的用法：

```
PS> Get-Help Get-Command
```

停止目前正在运行的以 'p' 字符开头来命名的所有程序：

```
PS> get-process p* | stop-process
```

停止目前正在运行的使用大于 1000MB 存储器的所有程序：

```
PS> get-process | where { $_.WS -gt 1000MB } | stop-process
```

计算某个目录下文件内的字节大小：

```
PS> get-childitem | measure-object -property length -sum
```

等待一个名为 “notepad” 的程序运行退出：

```
PS> $processToWatch = get-process notepad
```

```
PS> $processToWatch.WaitForExit()
```

将 "hello,world!" 字符串转为英文大写字符，使其变成 "HELLO,WORLD!"：

```
PS> "hello, world!".ToUpper()
```

在字符串 "string" 的第 1 个字符后插入字符串 "ABC"，使其变成 "sABCtring"：

```
PS> "string".Insert(1, "ABC")
```

订阅一个指定的 RSS Feed 并显示它最近的 8 个主题：

```
PS> $rssUrl = "http://blogs.msdn.com/powershell/rss.aspx"
```

```
PS> $blog = [xml](new-object System.Net.WebClient).DownloadString($rssUrl)
```

```
PS> $blog.rss.channel.item | select title -first 8
```

将 "\$UserProfile" 设置成数值 "UserProfile" 的环境变量：

```
PS> $UserProfile = $env:UserProfile
```



提示 更多的 Windows PowerShell 资料可查阅如下网址：

<http://www.pstips.net/powershell-online-tutorials>

[http://msdn.microsoft.com/en-us/library/dd835506\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/dd835506(VS.85).aspx)

## 4.2 Linux 基础

Linux 最初是作为支持英特尔 x86 架构的个人计算机的一个自由操作系统，目前 Linux 已经被移植到更多的计算机硬件平台上，远远超出其他的任何操作系统。Linux 发行版一直被用来作为服务器的主流操作系统，并且已经在该领域中占据了重要的地位，世界上 500 个最快的超级计算机 90% 以上运行的均是 Linux 发行版或其变种，包括最快的前 10 名超级计算机运行的都是基于 Linux 内核的操作系统。曾经是世界上最强大的超级计算机——IBM 的红杉（IBM Sequoia），已于 2011 年交付劳伦斯利福摩尔国家实验室，并于 2012 年 6 月开始运作，也是选择 Linux 作为操作系统。Linux 也广泛应用于嵌入式系统上，如手机、平板电脑、路由器、电视和电子游戏机等。广泛使用于移动设备上的 Android 操作系统就是创建于

Linux 内核之上的。4.2.1 节将以常用的 Linux 版本 CentOS 为例，来讲解 Linux 命令。

## 4.2.1 Linux 命令

### 1. Linux 命令基础

Linux 命令是对 Linux 系统进行管理的命令。对于 Linux 系统来说，无论是中央处理器、内存、磁盘驱动器、键盘、鼠标，还是用户等都是文件，Linux 系统管理的命令是它正常运行的核心，与之前的 DOS 命令类似。Linux 命令在系统中有两种类型：内置 Shell 命令和 Linux 命令。本节讲解 Linux 命令，Shell 命令将在 4.2.2 节讲解。

Linux 命令通常具有以下格式：

命令名 [ 命令选项 ] [ 命令参数 ]

先来看一个最简单的命令“su”，该命令用于用户的身份转换。su 命令可以转换用户身份，su 用户名表示转到某用户，若 su 命令不带参数则表示直接转到超级用户 root。命令提示符“\$”表示普通用户状态，“#”表示超级用户状态。下面的操作将演示身份转换：

```
$
$ su
```

密码：

```
#
```

### 2. 常用 Linux 命令

以下是系统信息与系统管理中常用的 Linux 命令的示例：

# arch	显示机器的处理器架构
# cal 2007	显示 2007 年的日历表
# cat /proc/cpuinfo	显示 CPU info 的信息
# cat /proc/interrupts	显示中断
# cat /proc/meminfo	校验内存使用
# cat /proc/swaps	显示哪些 swap 被使用
# cat /proc/version	显示内核的版本
# cat /proc/net/dev	显示网络适配器及统计
# cat /proc/mounts	显示已加载的文件系统
# clock -w	将对时间的修改保存到 BIOS
# date	显示系统日期
# lsusb -tv	显示 USB 设备
# uname -m	显示机器的处理器架构
# uname -r	显示正在使用的内核版本
# logout	注销
# reboot	重启
# shutdown -h now	关闭系统
# shutdown -h 22:50 &	按预定时间 22:50 关闭系统
# shutdown -c	取消按预定时间关闭系统
# shutdown -r now	重启

以下是文件和目录管理中常用的 Linux 命令：

# cd /home	进入 /home 目录
# cp file1 file2	复制一个文件
# ls	查看目录中的文件
# mkdir dir	创建 dir 目录
# mv mydir new_mydir	重命名 / 移动一个目录
# pwd	显示工作路径
# rm -f file	删除 file 文件
# find / -name myfile	从 '/' 开始进入根文件系统搜索 myfile 文件
# mount -o loop myfile.iso /mnt/cdrom	挂载一个文件或 ISO 镜像文件
# df -h	显示已经挂载的分区列表
# cat file1	从第一个字节开始正向查看文件的内容

以下是用户与用户组管理中常用的 Linux 命令:

# groupadd [group]	创建一个新用户组
# groupdel [group]	删除一个用户组
# passwd	修改口令
# passwd myuser	修改用户的口令
# useradd myuser	创建一个新用户
# userdel myuser	删除一个用户
# chgrp mygroup myfile	改变文件的群组

### 3. 主要命令解析及技巧

#### (1) ls 命令

ls 命令将指定目录的文件及目录输出在屏幕中。示例如下:

```
$ ls
hadoop-2.4.1  hadoop-2.4.1-src.tar.gz  hadoop-2.4.1.tar.gz  numpy  pypy-2.3.1-src
```

此外,通过加上“-la”参数可表示:以长格式的形式查看当前目录下的所有文件,包括隐藏文件。文件各字段的含义如下:

- 文件属性: drwxr-xr-x
- 文件硬链接数或目录子目录数: 3 (一个空目录的该字段是 2, 表示该目录下有两个子目录, 因为每一个目录都有一个指向它本身的子目录“.”和指向它上级目录的子目录“..”)
- 所有者: user
- 所属用户组: group
- 文件大小: 102 byte
- 修改时间: Mar11 22:56
- 文件名: Filename

下例演示了该参数的用法:

```
$ ls -la ~
总用量 150484
drwx-----. 6 myhaspl myhaspl 4096 9月 10 16:55 .
drwxr-xr-x. 3 root root 20 9月 10 08:23 ..
-rw-----. 1 myhaspl myhaspl 220 9月 10 17:59 .bash_history
```



```
drwxr-xr-x. 9 myhaspl myhaspl 4096 6月 21 14:38 hadoop-2.4.1
-rw-r--r--. 1 myhaspl myhaspl 15417097 6月 21 14:42 hadoop-2.4.1-src.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 138656756 6月 21 14:42 hadoop-2.4.1.tar.gz
drwxr-xr-x. 8 myhaspl myhaspl 4096 9月 10 17:02 numpy
drwxr-----. 3 myhaspl myhaspl 18 9月 10 16:21 .pki
drwxrwxr-x. 14 root root 4096 9月 10 16:25 pypy-2.3.1-src
```

## (2) cd 命令及多条命令同行

cd 命令的功能是切换当前目录，示例如下：

```
$cd ..
$cd /home/
$cd /opt
```

此外，还可以将多个命令写在同一行，一次使用，用分号隔开，比如：

```
$ ls -la ;cd numpy;ls
总用量 150484
drwx-----. 6 myhaspl myhaspl 4096 9月 10 16:55 .
drwxr-xr-x. 3 root root 20 9月 10 08:23 ..
-rw-----. 1 myhaspl myhaspl 220 9月 10 17:59 .bash_history
drwxr-xr-x. 9 myhaspl myhaspl 4096 6月 21 14:38 hadoop-2.4.1
-rw-r--r--. 1 myhaspl myhaspl 15417097 6月 21 14:42 hadoop-2.4.1-src.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 138656756 6月 21 14:42 hadoop-2.4.1.tar.gz
drwxr-xr-x. 8 myhaspl myhaspl 4096 9月 10 17:02 numpy
drwxr-----. 3 myhaspl myhaspl 18 9月 10 16:21 .pki
drwxrwxr-x. 14 root root 4096 9月 10 16:25 pypy-2.3.1-src
BENTO_BUILD.txt build INSTALL.txt pavement.py setupegg.py
THANKS.txt
bento.info COMPATIBILITY LICENSE.txt README.txt setup.py tools
branding DEV_README.txt MANIFEST.in release.sh site.cfg.example tox.ini
bscript doc numpy runtests.py TEST_COMMIT
$
```

如上面的结果所示，首先，先执行第1条ls命令，输出当前目录下的文件列表，然后，执行第2条cd命令，进入numpy目录后，执行第3条ls命令，输出numpy目录下的内容。

## (3) 命令后台执行

可以让进程在后台运行，执行命令后立即返回，这样还可以继续执行其他命令，只要在命令行的最后加上“&”即可。下面的示例是查找含有字符串“doc”的文件，因查找过程比较漫长，故而直接在后台执行并返回，如下所示：

```
$ find ~ -name doc &
[1] 5453
$ /home/myhaspl/hadoop-2.4.1/share/doc
/home/myhaspl/pypy-2.3.1-src/site-packages/numpy/doc
/home/myhaspl/pypy-2.3.1-src/ctypes_configure/doc
/home/myhaspl/pypy-2.3.1-src/pypy/doc
/home/myhaspl/numpy/doc
/home/myhaspl/numpy/build/lib.linux-x86_64-2.7/numpy/doc
/home/myhaspl/numpy/numpy/doc
```



```
/home/myhaspl/numpy/numpy/numpy/doc
/home/myhaspl/numpy/numpy/numpy/f2py/doc
```

#### (4) 重定向与管道

重定向是指对原来系统命令的默认执行方式进行改变，比如将原本在显示器中的输出改为输出到某一文件中。常用的重定向命令如下所示（cmd 表示命令，file 表示文件或设备）：

```
cmd > file      把 stdout 重定向到 file 文件中。
cmd >> file     把 stdout 重定向到 file 文件中（追加）。
cmd 1> file     把 stdout 重定向到 file 文件中。
cmd > file 2>&1  把 stdout 和 stderr 一起重定向到 file 文件中。
cmd 2> file     把 stderr 重定向到 file 文件中。
cmd 2>> file    把 stderr 重定向到 file 文件中（追加）。
```

下面以实例来讲解主要的重定向操作。首先讲解重定向操作符“>”，下面的例子演示了先使用 find 命令来查找含有“doc”的文件，然后使用“>”将结果列表重定向到一个文件中：

```
$ find ~ -name doc >mydoclist &
[1] 5461
$ ls
hadoop-2.4.1          hadoop-2.4.1.tar.gz  numpy
hadoop-2.4.1-src.tar.gz mydoclist            pypy-2.3.1-src
[1]+  完成                  find ~ -name doc > mydoclist
$ cat mydoclist
/home/myhaspl/hadoop-2.4.1/share/doc
/home/myhaspl/pypy-2.3.1-src/site-packages/numpy/doc
/home/myhaspl/pypy-2.3.1-src/ctypes_configure/doc
/home/myhaspl/pypy-2.3.1-src/pypy/doc
/home/myhaspl/numpy/doc
/home/myhaspl/numpy/build/lib.linux-x86_64-2.7/numpy/doc
/home/myhaspl/numpy/numpy/doc
/home/myhaspl/numpy/numpy/numpy/doc
/home/myhaspl/numpy/numpy/numpy/f2py/doc
/home/myhaspl/numpy/numpy/build/lib.linux-x86_64-2.7/numpy/doc
$
```

接下来讲解重定向操作符“>>”，下面的例子先使用 find 命令查找文件，然后使用“>>”将查找结果重定向到文件 mydoclist 中：

```
$ find ~ -name hadoop >>mydoclist &
[1] 5466
$ cat mydoclist
/home/myhaspl/hadoop-2.4.1/share/doc
/home/myhaspl/pypy-2.3.1-src/site-packages/numpy/doc
/home/myhaspl/pypy-2.3.1-src/ctypes_configure/doc
/home/myhaspl/pypy-2.3.1-src/pypy/doc
/home/myhaspl/numpy/doc
/home/myhaspl/numpy/build/lib.linux-x86_64-2.7/numpy/doc
```

```

/home/myhaspl/numpy/numpy/doc
/home/myhaspl/numpy/numpy/numpy/doc
/home/myhaspl/numpy/numpy/f2py/doc
/home/myhaspl/numpy/numpy/build/lib.linux-x86_64-2.7/numpy/doc
/home/myhaspl/hadoop-2.4.1/bin/hadoop
/home/myhaspl/hadoop-2.4.1/etc/hadoop
/home/myhaspl/hadoop-2.4.1/share/hadoop
/home/myhaspl/hadoop-2.4.1/share/hadoop/httpfs/tomcat/webapps/webhdfs/WEB-INF/
classes/org/apache/hadoop
/home/myhaspl/hadoop-2.4.1/share/hadoop/httpfs/tomcat/webapps/webhdfs/WEB-INF/
classes/org/apache/hadoop/lib/service/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/api/src-html/org/apache/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/api/org/apache/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/api/org/apache/hadoop/lib/service/
hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/hadoop-hdfs-httpfs/apidocs/src-
html/org/apache/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/hadoop-hdfs-httpfs/apidocs/src-
html/org/apache/hadoop/lib/service/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/hadoop-hdfs-httpfs/apidocs/org/
apache/hadoop
/home/myhaspl/hadoop-2.4.1/share/doc/hadoop/hadoop-hdfs-httpfs/apidocs/org/
apache/hadoop/lib/service/hadoop

```

再接下来，看一个 I/O 重定向的例子。在 Linux 系统中，标准输入（stdin）的文件描述符为 0，标准输出（stdout）的文件描述符为 1，标准错误输出（stderr）的文件描述符为 2。

标准输出重定向命令格式如下所示：

```
1>filename 或 1>>filename
```

下例演示了 I/O 重定向的使用方法，将标准输出重定向到文件 abc 中：

```

$ echo "aaa" 1> abc
$ cat abc
aaa
$ echo "aaa" 1>> abc
$ cat abc
aaa
aaa
$

```

标准错误输出重定向命令格式如下所示：

```
2>filename 或 2>>filename
```

下例演示了如何将标准错误输出重定向到文件 error.log 中：

```

$ rm /root/* 2>error.log
$ cat error.log
rm: 无法删除 "/root/*": 权限不够
$

```

此外,还可以使用“i>j”将文件描述符 i 表示的输出文件重定向到文件描述符 j 表示的文件中。

最后,讲解 tee 命令与管道操作符“|”的合并使用,实现在输出的同时,再输出一份同样的内容给管道。下例演示了如何列出目录的内容并输出给 mylist 文件:

```
$ ls |tee mylist
hadoop-2.4.1
hadoop-2.4.1-src.tar.gz
hadoop-2.4.1.tar.gz
mydoclist
numpy
pypy-2.3.1-src
$ cat mylist
hadoop-2.4.1
hadoop-2.4.1-src.tar.gz
hadoop-2.4.1.tar.gz
mydoclist
numpy
pypy-2.3.1-src
$
```



**提示** 管道是 Linux 中很重要的一种通信方式,是将一个程序的输出直接连接到另一个程序的输入。

### (5) 通配符

在文件管理等操作中,可以使用通配符,最常用的通配符如下:

□ \*: 多个字符

□ ?: 单个字符

此外,还可以使用“~”来表示当前用户的主目录。

下例演示了如何使用 ls 命令加通配符的模式列出指定目录“~/numpy/”的内容:

```
$ ls ~/numpy/*.py
/home/myhaspl/numpy/pavement.py /home/myhaspl/numpy/setupegg.py
/home/myhaspl/numpy/runtests.py /home/myhaspl/numpy/setup.py
$ ls ~/numpy/*.txt
/home/myhaspl/numpy/BENTO_BUILD.txt /home/myhaspl/numpy/LICENSE.txt
/home/myhaspl/numpy/DEV_README.txt /home/myhaspl/numpy/README.txt
/home/myhaspl/numpy/INSTALL.txt /home/myhaspl/numpy/THANKS.txt
$ ls ~/numpy/setup*.py
/home/myhaspl/numpy/setupegg.py /home/myhaspl/numpy/setup.py
$ ls ~/numpy/setup.??
/home/myhaspl/numpy/setup.py
$
```

### (6) 作业管理

可以使用 jobs 命令显示当前作业, grep 命令是一种强大的文本搜索工具,它能使用正则

表达式搜索文本。下例将这两者结合起来,输出文件列表中包括字符串"se"的文件名,并将输出结果放到 myse 文件中:

```
$ find ~ -name "*.py"|grep se >myse &
[1] 2258
$ jobs
[1]+ 运行中                  find ~ -name "*.py" | grep se > myse &
$ jobs
[1]+ 完成                    find ~ -name "*.py" | grep se > myse
$ cat myse
/home/myhaspl/pypy-2.3.1-src/dotviewer/graphparse.py
/home/myhaspl/pypy-2.3.1-src/dotviewer/graphserver.py
```

下例演示了使用 kill 命令来停止作业,其中 sleep 表示该作业休眠,后面的时间参数可以是 s(秒)、h(小时)、m(分钟)或 d(日数):

```
$ (find ~ -name ".*y"|grep se >myse;sleep 10s )&
[1] 2365
$ jobs
[1]+ 运行中                  ( find ~ -name ".*y" | grep --color=auto se > myse; sleep 10s ) &
$ kill %1
$ jobs
[1]+ 已终止                  ( find ~ -name ".*y" | grep --color=auto se > myse; sleep 10s )
$
```

### (7) 查询命令使用方式

可以使用 man 命令行的方式来查询命令帮助,下面演示了查询 ls 命令的帮助信息:

```
man ls
```

目录:

□ ~ 表示当前用户的主目录。

□ . 表示当前目录。

□ .. 表示上级目录。

### (8) 链接文件

在 Linux 中,可用不同的文件名引用同一个数据或程序,称为硬链接,可在同一物理文件系统中,创建硬链接。下例演示了硬链接的使用方法:

```
$ ls -la
总用量 151228
drwx-----. 6 myhaspl myhaspl      4096 9月 18 08:55 .
drwxr-xr-x. 3 root root              20 9月 10 08:23 ..
-rw-----. 1 myhaspl myhaspl      1915 9月 16 18:05 .bash_history
drwxr-xr-x. 9 myhaspl myhaspl      4096 6月 21 14:38 hadoop-2.4.1
-rw-r--r--. 1 myhaspl myhaspl 15417097 6月 21 14:42 hadoop-2.4.1-src.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 138656756 6月 21 14:42 hadoop-2.4.1.tar.gz
-rw-r--r--. 1 myhaspl myhaspl      1454 9月 16 10:53 mydoclist
-rw-rw-r--. 1 myhaspl myhaspl         88 9月 16 17:25 mylist
-rw-rw-r--. 1 myhaspl myhaspl    357304 9月 18 08:55 mypylist
```

```

-rw-rw-r--. 1 myhaspl myhaspl 31954 9月 16 18:02 myse
drwxr-xr-x. 8 myhaspl myhaspl 4096 9月 16 10:39 numpy
drwxr-----. 3 myhaspl myhaspl 18 9月 10 16:21 .pki
drwxrwxr-x. 14 root root 4096 9月 10 16:25 pypy-2.3.1-src
-rw-rw-r--. 1 myhaspl myhaspl 357304 9月 16 17:57 se
$ ln mypylist mypylist1
$ ls -la
总用量 151580
drwx-----. 6 myhaspl myhaspl 4096 9月 18 08:56 .
drwxr-xr-x. 3 root root 20 9月 10 08:23 ..
-rw-----. 1 myhaspl myhaspl 1915 9月 16 18:05 .bash_history
drwxr-xr-x. 9 myhaspl myhaspl 4096 6月 21 14:38 hadoop-2.4.1
-rw-r--r--. 1 myhaspl myhaspl 15417097 6月 21 14:42 hadoop-2.4.1-src.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 138656756 6月 21 14:42 hadoop-2.4.1.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 1454 9月 16 10:53 mydoclist
-rw-rw-r--. 1 myhaspl myhaspl 88 9月 16 17:25 mylist
-rw-rw-r--. 2 myhaspl myhaspl 357304 9月 18 08:55 mypylist
-rw-rw-r--. 2 myhaspl myhaspl 357304 9月 18 08:55 mypylist1
-rw-rw-r--. 1 myhaspl myhaspl 31954 9月 16 18:02 myse
drwxr-xr-x. 8 myhaspl myhaspl 4096 9月 16 10:39 numpy
drwxr-----. 3 myhaspl myhaspl 18 9月 10 16:21 .pki
drwxrwxr-x. 14 root root 4096 9月 10 16:25 pypy-2.3.1-src
-rw-rw-r--. 1 myhaspl myhaspl 357304 9月 16 17:57 se
$ ln mypylist mypylist2
$ ls -la
总用量 151932
drwx-----. 6 myhaspl myhaspl 4096 9月 18 09:27 .
drwxr-xr-x. 3 root root 20 9月 10 08:23 ..
-rw-----. 1 myhaspl myhaspl 1915 9月 16 18:05 .bash_history
drwxr-xr-x. 9 myhaspl myhaspl 4096 6月 21 14:38 hadoop-2.4.1
-rw-r--r--. 1 myhaspl myhaspl 15417097 6月 21 14:42 hadoop-2.4.1-src.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 138656756 6月 21 14:42 hadoop-2.4.1.tar.gz
-rw-r--r--. 1 myhaspl myhaspl 1454 9月 16 10:53 mydoclist
-rw-rw-r--. 1 myhaspl myhaspl 88 9月 16 17:25 mylist
-rw-rw-r--. 3 myhaspl myhaspl 357304 9月 18 08:55 mypylist
-rw-rw-r--. 3 myhaspl myhaspl 357304 9月 18 08:55 mypylist1
-rw-rw-r--. 3 myhaspl myhaspl 357304 9月 18 08:55 mypylist2
-rw-rw-r--. 1 myhaspl myhaspl 31954 9月 16 18:02 myse
drwxr-xr-x. 8 myhaspl myhaspl 4096 9月 16 10:39 numpy
drwxr-----. 3 myhaspl myhaspl 18 9月 10 16:21 .pki
drwxrwxr-x. 14 root root 4096 9月 10 16:25 pypy-2.3.1-src
-rw-rw-r--. 1 myhaspl myhaspl 357304 9月 16 17:57 se

```

从几条命令的执行结果可以看到：mypylist 的硬链接数量在增加。

在 Linux 下也可以创建软链接，这种链接跨越了不同的物理文件系统，也称为符号链接文件，与硬链接不同的是，它是一个单独的文件，存放着目标文件的路径名。下例演示了软链接的使用方式：

```

$ ln -s mypylist mypylists
$ ls -la
总用量 151932
drwx-----. 6 myhaspl myhaspl 4096 9月 18 09:35 .

```



```

drwxr-xr-x. 3 root root 20 9月 10 08:23 ..
-rw----- 1 myhaspl myhaspl 1915 9月 16 18:05 .bash_history
drwxr-xr-x. 9 myhaspl myhaspl 4096 6月 21 14:38 hadoop-2.4.1
-rw-r--r-- 1 myhaspl myhaspl 15417097 6月 21 14:42 hadoop-2.4.1-src.tar.gz
-rw-r--r-- 1 myhaspl myhaspl 138656756 6月 21 14:42 hadoop-2.4.1.tar.gz
-rw-r--r-- 1 myhaspl myhaspl 1454 9月 16 10:53 mydoclist
-rw-rw-r-- 1 myhaspl myhaspl 88 9月 16 17:25 mylist
-rw-rw-r-- 3 myhaspl myhaspl 357304 9月 18 08:55 mypylist
-rw-rw-r-- 3 myhaspl myhaspl 357304 9月 18 08:55 mypylist1
-rw-rw-r-- 3 myhaspl myhaspl 357304 9月 18 08:55 mypylist2
lrwxrwxrwx. 1 myhaspl myhaspl 8 9月 18 09:35 mypylists -> mypylist

```

### (9) 文件权限

对于一般的 Linux 文件而言，权限如下所示：

□ r：允许读文件内容。

□ w：允许修改文件内容。

□ x：允许执行该文件。

对于 Linux 目录而言，权限如下所示：

□ r：允许列出该目录下的文件和子目录。

□ w：允许生成和删除该目录下的文件。

□ x：允许访问该目录。

□ u：代表所有者 (user)。

□ g：代表所有者所在的组群 (group)。

□ o：代表其他人，但不是 u 和 g (other)。

□ a：代表全部的人，也就是包括 u、g 和 o。

此外，还可以通过 `chmod` 命令来改变权限。该命令的格式如下所示：

`chmod [用户类型](+/-) 访问权限的格式文件或目录名`

下例演示了如何改变权限，将 `mytext` 设置为所有的人可写为：

```
chmod a+w mytext
```

下例演示了文件 `myrun` 的生成、权限的授予及最后执行的过程：

```

$ echo "ls;echo \"ok\"" >myrun
$ cat myrun
ls;echo "ok"
$ chmod +x ./myrun
$ ./myrun
abc      hadoop-2.4.1      mydoclist  mypylist1  myrun  pypy-2.3.1-src
abd      hadoop-2.4.1-src.tar.gz  mylist    mypylist2  myse    se
error.log hadoop-2.4.1.tar.gz      mypylist  mypylists  numpy
ok

```

### (10) 进程管理

可通过 `ps` 命令显示当前进程，通过 `kill` 命令杀死进程，`kill` 命令终止进程的调用格式如

下 (pid 为进程号):

```
kill -9 pid
```

下例演示了显示进程及杀死进程的过程：首先，执行 ls 命令后开始休眠，整个过程处于一个进程内，然后在该进程休眠后，通过 ps 命令查找该进程的 PID 号，最后用 kill 命令终止该进程。

```
$ ls;sleep 20&
$ ps -a
  PID TTY          TIME CMD
 2245 pts/0    00:00:00 sleep
 2246 pts/0    00:00:00 ps
$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1         0  0  16:07 ?            00:00:01 /usr/lib/systemd/systemd
--switched-root --
root           2         0  0  16:07 ?            00:00:00 [kthreadd]
root           3         2  0  16:07 ?            00:00:00 [ksoftirqd/0]
root           5         2  0  16:07 ?            00:00:00 [kworker/0:0H]
root           6         2  0  16:07 ?            00:00:00 [kworker/u2:0]
.....
myhaspl    2245    2176   0 17:04 pts/0    00:00:00 sleep 20
myhaspl    2247    2176   0 17:04 pts/0    00:00:00 ps -ef
$ kill -9 2287
myhaspl    2289    2261   0 17:11 pts/1    00:00:00 ps -ef
[1]+  已杀死                  sleep 20
$
```

(11) 用户管理

下例演示了通过 useradd 命令、userdel 命令进行增加或删除用户的操作：

```
# useradd -m test1
# ls /home/
myhaspl test1
# su test1
$ su
# userdel test1
```

(12) 磁盘空间管理

下例演示了通过 df 命令来查看空间的使用情况：

```
# df
文件系统              1K- 块      已用      可用      已用 %      挂载点
/dev/mapper/centos-root 7022592 2545964 4476628    37%      /
devtmpfs                632192         0  632192     0%      /dev
tmpfs                   638148         0  638148     0%      /dev/shm
tmpfs                   638148    8320  629828     2%      /run
tmpfs                   638148         0  638148     0%      /sys/fs/cgroup
/dev/sda1                508588 126640  381948    25%      /boot
#
```

下例演示了通过 du 命令来查看某个目录或文件的占用空间：

```
$ du -h myrun
4.0K    myrun
$ du -h numpy
4.0K    numpy/.git/refs/heads
0       numpy/.git/refs/tags
4.0K    numpy/.git/refs/remotes/origin
4.0K    numpy/.git/refs/remotes
8.0K    numpy/.git/refs
0       numpy/.git/branches
44K     numpy/.git/hooks
4.0K    numpy/.git/info
28M     numpy/.git/objects/pack
```

## 4.2.2 Shell 基础

Shell 是 Linux 系统的用户界面，为用户与内核进行交互操作提供的一种接口。它接收用户输入的命令并把它送入内核执行。实际上 Shell 是一个命令解释器，它解释用户输入的命令并且把它们送到内核中。不仅如此，Shell 还有自己的编程语言，用于对命令进行编辑，它允许用户编写由 Shell 命令组成的程序。Shell 编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的 Shell 程序与其他应用程序具有同样的效果。

### 1. 建立和运行 Shell 文件

什么是 Shell 程序呢？简单地说 Shell 程序就是一个包含若干行 Shell 或 Linux 命令的文件。下面以实例来说明如何建立并运行 Shell 文件。

首先编辑如下 Shell 文件 test1.sh，并将扩展名命名为 ".sh"：

```
#!/bin/sh
ls -la
cd numpy
ls
```

然后，对 Shell 文件进行权限授权，如下所示：

```
$ chmod a+rx test1.sh
```

最后运行该 Shell 文件，如下所示：

```
$ ./test1.sh
```

### 2. Shell 的命令行参数

1) 读取某个命令行参数。可使用 \$0 到 \$n 表示对第 0 到第 n 个参数进行操作。下面演示了依次输出程序的命令行参数：

```
$ cat test1.sh
#!/bin/sh
```

```

echo "$0 "
echo "$1 "
echo "$2 "
$ ./test1.sh a b c
./test1.sh
a
b

```

2) 弹出所有命令行参数。shift 从命令行参数中弹出第 1 个参数，until 开始循环，如下所示：

```

$ cat test1.sh
#!/bin/sh
until [ -z "$1" ]
do
    echo "$1 "
    shift
done
$ ./test1.sh a b c d e f
a
b
c
d
e
f

```

3) 使用 \$\* 和 \$@ 表示所有参数。下例演示了通过 for 循环依次读取命令行参数列表中的元素并输出：

```

$ cat test1.sh
#!/bin/sh
index=1
for myarg in $*
do
    echo "NO#$index=$myarg"
    let "index+=1"
done
$ ./test1.sh a b c d e f
NO#1=a
NO#2=b
NO#3=c
NO#4=d
NO#5=e
NO#6=f

```

### 3. Shell 变量

1) 读写变量。可以通过在变量名的前面加 “\$” 取变量的值或以 “\${变量名}” 的方式对变量进行操作。

下面例子首先定义了 a、b 两个变量，然后将 a 变量的值赋值给 b 变量，最后输出 b 变量：

```

$ cat test1.sh

```

```
#!/bin/sh
a=12
b=$a
echo $b
$ ./test1.sh
12
$
```

2) 变量的间接引用。下例演示了通过“\${!变量名}”的方式对变量进行间接引用，读取变量值：

```
$ cat test1.sh
#!/bin/sh
a=12
b=a
echo ${!b}
$ ./test1.sh
12
$
```

#### 4. 条件表达式

1) 下面的例子演示了如何通过“if...then...else..fi”的方式来完成条件选择，如果 a>b，则输出 GT，否则输出 LT：

```
$ cat test1.sh
#!/bin/sh
a=1
b=2
if [ $a -gt $b ]
then
    echo "GT"
else
    echo "LT"
fi
$ ./test1.sh
LT
$
```

2) 下面的例子演示了如何通过“if...then.. elif ...then...else..fi”的方式来完成条件选择，如果 a>b，则输出 GT，如果 a==b 则输出 eq，否则输出 LT：

```
$ cat test1.sh
#!/bin/sh
a=2
b=2
if [ $a -gt $b ]
then
    echo "GT"
elif [ $a -eq $b ]
then
    echo "eq"
fi
```



```
else
    echo "LT"
```

```
fi
```

```
$ ./test1.sh
```

```
eq
```

```
$
```

3) 下面的例子演示了如何通过“case...in...esac...”语句来完成条件选择，程序输出菜单后，用户选择适合的菜单项，然后通过 case 系列语句读取用户的选择并输出：

```
$ cat test1.sh
```

```
#!/bin/sh
```

```
echo "=====
```

```
echo "1.a"
```

```
echo "2.b"
```

```
echo "3.c"
```

```
read mychoice
```

```
case $mychoice in
```

```
1 ) echo "a";;
```

```
2 ) echo "b";;
```

```
3 ) echo "c";;
```

```
esac
```

```
exit 0
```

```
$ ./test1.sh
```

```
=====
```

```
1.a
```

```
2.b
```

```
3.c
```

```
2
```

```
b
```

```
$
```

## 5. 循环

### (1) for 循环

下例演示了如何通过“for...in...do...done”的语句来完成循环的操作：程序首先执行 ls 命令，读取当前目录下的文件列表，然后使用 for 语句逐个读取文件列表并输出文件名。

```
$ cat test1.sh
```

```
#!/bin/sh
```

```
for filename in `ls`
```

```
do
```

```
    echo $filename
```

```
done
```

```
$ ./test1.sh
```

```
1
```

```
abc
```

```
abd
```

```
error.log
```

```
hadoop-2.4.1
```

```
hadoop-2.4.1-src.tar.gz
```

```
hadoop-2.4.1.tar.gz
```

```
hello
mydoclist
```

下例演示了如何列出非目录性质的文件的操作：程序首先执行 `ls` 命令，读取当前目录下的文件列表，然后使用 `for` 语句逐个读取文件列表，对列表中的每个文件使用 “`-f`” 进行检测，如果是文件，则输出文件名，否则跳过不处理。

```
$ cat test1.sh
#!/bin/sh
for filename in `ls`
do
    if [ -f $filename ]
    then
        echo $filename
    fi
done
$ ./test1.sh
1
abc
abd
error.log
hadoop-2.4.1-src.tar.gz
hadoop-2.4.1.tar.gz
hello
mydoclist
myl2
mylist
mypylst
mypylst1
mypylst2
myrun
myse
se
test1.sh
$
```

## (2) while 循环

下例演示了 `while` 循环的操作，程序通过 `while` 循环来完成将变量 `a` 的值由 1 递增到 9，并输出：

```
$ cat test1.sh
#!/bin/sh
a=1
while [ $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
$ ./test1.sh
1
2
3
```

```

4
5
6
7
8
9
$

```

下例演示了如何使用 while 循环来显示所有的偶数，其中，将条件设为 “:” 或 “true”，表示恒为真的意思：

```

$ cat test1.sh
#!/bin/sh
a=0
while :
do
    let "a=$a + 1"
    if [ $a -gt 20 ]
    then
        break
    fi
    if [ $((($a%2)) -eq 1 ]
    then
        continue
    fi
    echo $a
done
$ ./test1.sh
2
4
6
8
10
12
14
16
18
20
$

```

下例演示了如何使用 while 循环来实现重定向 I/O 的功能：

```

$ cat test1.sh
#!/bin/sh
while read name age
do
    echo $name
    echo $age
done<student.txt
$ cat student.txt
zhangsan 20
lisi 18
liumi 19

```

```
$ ./test1.sh
zhangsan
20
lisi
18
liumi
19
$
```

### (3) until 循环

下例演示了如何通过 until 循环来完成变量 a 的递增，并在满足条件 (a>10) 后退出：

```
$ cat test1.sh
#!/bin/sh
a=1
until [ $a -gt 10 ]
do
    echo $a
    a='expr $a + 1'
done
$ ./test1.sh
1
2
3
4
5
6
7
8
9
10
$
```

下例演示了如何通过 until 循环来进行重定向：

```
$ cat test1.sh
#!/bin/sh
until ! read name age
do
    echo $name
    echo $age
done<student.txt
$ ./test1.sh
zhangsan
20
lisi
18
liumi
19
```

## 6. select 菜单

select 使用 PS3 环境变量的值作为提示符，下例演示了如何使用 select 来实现类似菜单

的功能,用户在年龄为“<18”、“<28”和“<60”之间选择:

```
$ cat test1.sh
#!/bin/sh
PS3="choice:"
echo
select age in "<18" "<28" "<60"
do
    echo
    echo "age is $age"
    break
done
exit 0
$ ./test1.sh
1) <18
2) <28
3) <60
choice:2
age is <28
$
```

until 的重定向示例如下:

```
$ cat test1.sh
#!/bin/sh
until ! read name age
do
    echo $name
    echo $age
done<student.txt
$ ./test1.sh
zhangsan
20
lisi
18
liumi
19
$
```

## 7. Shell 函数

下例定义了函数 isodd, 用于判断奇偶数:

```
$ cat test1.sh
#!/bin/sh
isodd(){
    if [ ((${1%2}) -eq 1 ]
    then
        return 1
    else
        return 0
    fi
}
```



```

read mynum
isodd $mynum
myodd=$?
if [ $myodd -eq 1 ]
then
    echo "$mynum is an odd number"
else
    echo "$mynum is an even number"
fi
$ ./test1.sh
12
12 is an even number
$ ./test1.sh
33
33 is an odd number

```

## 4.3 Vim 编辑器

### 4.3.1 Vim 编辑器概述

Vim 是从 Vi 发展出来的一个文本编辑器。代码补充、编译及错误跳转等便于编程的功能特别丰富，在程序员中被广泛使用。与 Emacs 并列成为类 UNIX 系统用户最喜欢的编辑器。Vim 的操作模式很特别，它强大的编辑能力中有很很大一部分是来自于模式和命令的组合。

命令的组合是指通过一些非常简短的字符（包括英文、数字、符号等）组合成各种命令。比如：普通模式命令“dd”表示删除当前行，“dj”表示删除到下一行，原理是第一个“d”的含义是删除，“j”键表示移动到下一行，组合后“dj”表示删除当前行和下一行。另外还可以指定命令的重复次数，“2dd”（重复“dd”两次）和“dj”的效果是一样的。“d^”，其中“^”代表行首，故组合后的含义是删除从光标开始到行首间的内容（不包含光标）；“d\$”，其中“\$”代表行尾，删除从光标到行尾的内容（包含光标）；用户若学习了各种各样的文本间移动或跳转的命令和其他的普通模式的编辑命令，并且能够灵活地组合使用的话，那么就能够比那些没有模式的编辑器更加高效地进行文本编辑。

模式的组合是指通过一些非常简短的英语字符就可转换进入各种模式。比如：在普通模式中，有很多方法可以进入插入模式。比较普通的方式是按“a”（append/追加）键或“i”（insert/插入）键。

可通过下面的方式来启动 Vim：

```
$vim
```

启动后，显示界面如图 4-1 所示。

启动后可进行一些简单的操作。首先，可按 i 键进入插入模式，输入字符，如图 4-2 所示。

然后，按 Esc 键退出插入模式，输入“:wq! hello”，并以“hello”为文件名，存盘退出。

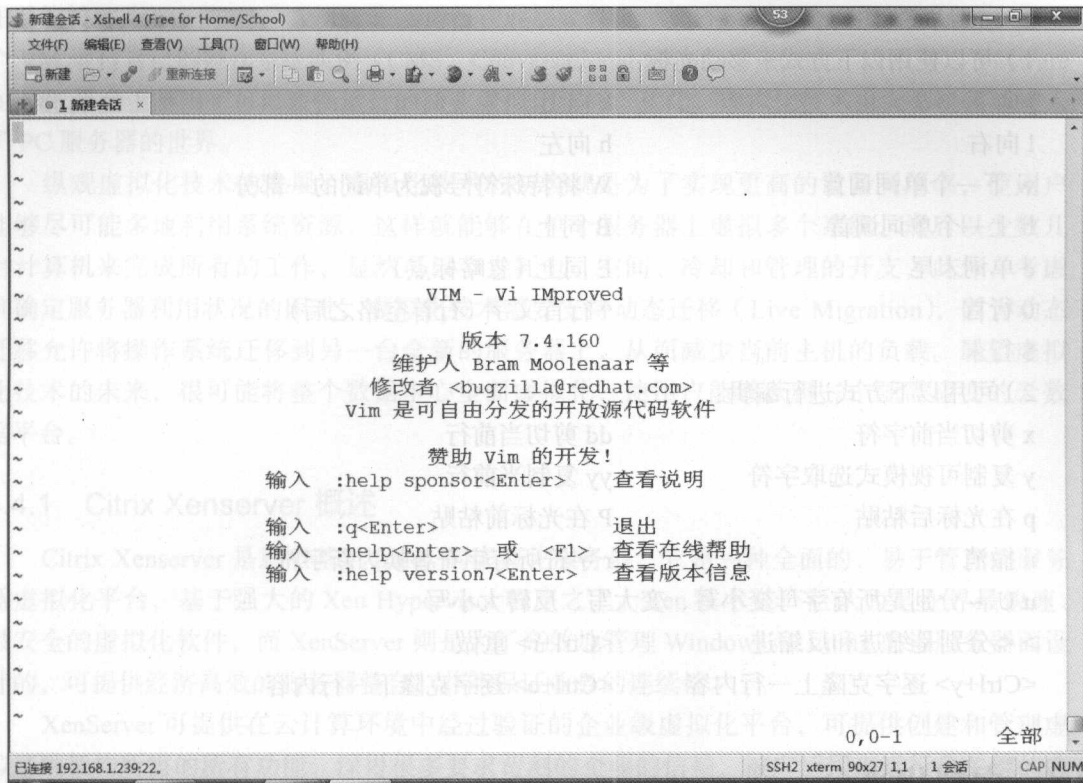


图 4-1 Vim 界面

最后，可通过 cat 命令显示文件内容，如下所示：

```
$ cat hello
hello
world!
```

### 4.3.2 Vim 常用命令

#### 1. 模式转换命令

模式间切换的方法有如下三种情况。

- ❑ 其他模式转普通模式：Esc 键。
- ❑ 普通模式转插入模式，如下列命令所示：

- |            |             |
|------------|-------------|
| i 在光标前插入   | I 在行首插入     |
| a 在光标后插入   | A 在行末插入     |
| o 在当行之下新建行 | O 在当行之上新建行  |
| r 替换当前字符   | R 从当前字符开始替换 |

- ❑ 普通模式转命令模式：按“:”键。

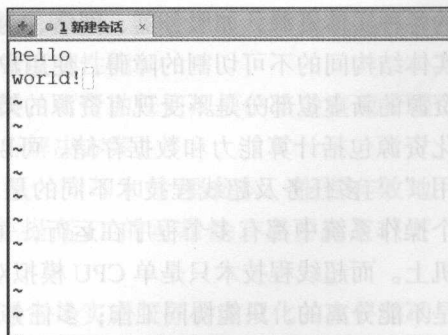


图 4-2 插入模式

## 2. 常用命令

1) 可以使用以下方式移动光标:

j 向下	k 向上
l 向右	h 向左
w 下一个单词词首	W 将特殊符号视为单词的一部分
b 上一个单词词首	B 同上
e 单词末尾	E 同上 (忽略标点)
0 行首	^ 行首文字 (行首空格之后)
\$ 行末	

2) 可用以下方式进行编辑:

x 剪切当前字符	dd 剪切当前行
y 复制可视模式选取字符	yy 复制当前行
p 在光标后粘贴	P 在光标前粘贴
u 撤消	r 字符所有字符替换为新字符
u U ~ 分别是所有字母变小写、变大写、反转大小写	
> < 分别是缩进和反缩进	<Ctrl+r> 重做
<Ctrl+y> 逐字克隆上一行内容	<Ctrl+e> 逐字克隆下一行内容

## 4.4 虚拟化平台

维基百科将虚拟化 (Virtualization) 定义为: “虚拟化是一种资源管理技术, 是将计算机的各种实体资源, 如服务器、网络、内存及存储等, 予以抽象、转换, 然后呈现出来, 打破实体结构间的不可切割的障碍, 使用户能以比原本的组态更好的方式来应用这些资源。这些资源的新虚拟部分是不受现有资源的架设方式、地域或物理组态所限制的, 一般所指的虚拟化资源包括计算能力和数据存储。可以看到它始终如一的目标就是实现对 IT 资源的充分利用。”与多任务及超线程技术不同的是, 虚拟化技术允许同时运行多个操作系统, 而且每一个操作系统中都有多个程序在运行, 每一个操作系统都运行在一个虚拟的 CPU 或是虚拟主机上。而超线程技术只是单 CPU 模拟双 CPU 来平衡程序运行性能, 这两个模拟出来的 CPU 是不能分离的, 只能协同工作; 多任务则是指在一个操作系统中多个程序同时一起运行。

1959 年, 克里斯托弗 (Christopher Strachey) 发表了一篇学术报告, 名为 “大型高速计算机中的时间共享”, 他在文中提出了虚拟化的基本概念, 这篇文章也被认为是虚拟化技术的最早论述。可以说虚拟化作为一个概念被正式提出就是从此时开始的。最早在商业系统上实现虚拟化的是 IBM 公司在 1965 年发布的 IBM 7044, 它允许用户在一台主机上运行多个操作系统, 让用户尽可能充分地利用昂贵的大型机资源, 之后 IBM 还开发了型号为 Model 67 的 System/360 主机, Model 67 主机通过虚拟机监视器 (Virtual Machine Monitor) 虚拟所有的硬件接口。在早期的计算中, 操作系统被称为 Supervisor, 能够运行在其他操作系统之上的操

作系统被称为 hypervisor, VMM 直接运行在底层硬件上, 允许执行多个虚拟机 (VM), 每一个 VM 运行自己的操作系统实例 (CMS, Conversational Monitor System)。1999 年, VMware 在 X86 平台上推出了可以流畅运行的商业虚拟化软件, 从此, 虚拟化技术由大型机领域进入了 PC 服务器的世界。

纵观虚拟化技术的发展, 虚拟化技术的初衷就是为了实现更高的设备利用率, 使用户能够尽可能多地利用系统资源, 这样就能够在单个服务器上虚拟多个系统, 然后以少数几台计算机来完成所有的工作, 显然可以节省耗电、空间、冷却和管理的开支, 此外, 考虑到确定服务器利用状况的困难, 虚拟化技术需要支持动态迁移 (Live Migration), 因为动态迁移允许将操作系统迁移到另一台全新的服务器上, 从而减少当前主机的负载。展望虚拟化技术的未来, 很可能将整个数据中心全面虚拟化, 使用户能够获得一个按需应变的云数据平台。

#### 4.4.1 Citrix XenServer 概述

Citrix XenServer 是思杰基于 Linux 的虚拟化服务器。它是一种全面的、易于管理的服务器虚拟化平台, 基于强大的 Xen Hypervisor 程序之上。Xen 技术被广泛认为是业界最快速、最安全的虚拟化软件, 而 XenServer 则是为了高效地管理 Windows 和 Linux 虚拟服务器而设计的, 可提供经济高效的服务器整合, 并能保证业务的连续性。

XenServer 可提供在云计算环境中经过验证的企业级虚拟化平台, 可提供创建和管理虚拟基础架构所需的所有功能, 深得很多要求苛刻的企业信赖, 被广泛用来运行最关键的应用, 而且被最大规模的云计算环境和 xSP 所采用。

XenServer 分为免费版和 Premium 版, 它们各自的特征如下:

- ❑ 免费版 XenServer 配备有 64 位系统管理程序和集中管理、实时迁移及转换工具, 可创建一个虚拟化平台来最大限度地提高虚拟机的密度和性能。
- ❑ Premium 版 XenServer 对平台进行了更深层的扩展, 可帮助任何规模的企业实现管理流程的集成和自动化, 是一种先进的虚拟数据中心解决方案。

XenServer 可整合服务器工作负载, 进而节约电源、冷却和管理成本, 能更有效地适应不断变化的 IT 环境, 优化利用现有的硬件设备并提高 IT 的可靠性, 主要拥有以下优势:

- ❑ 将 IT 成本降低 50% 甚至更多。虽然服务器整合通常是实施服务器虚拟化的主要驱动因素, 但企业可以获得更多的优势, 而不仅仅是服务器总数量的减少。XenServer 虚拟化管理工具可以将服务器的要求降低 10 倍。数据中心内的服务器整合可以降低功耗和管理成本, 同时还可以打造更绿色环保的 IT 环境。
- ❑ 提高 IT 灵活性。虚拟化使数据中心能够灵活地适应不断变化的 IT 要求。例如, XenServer 可以创建出能够无缝集成现有存储环境的虚拟基础架构。这样就可以缩短 IT 部门满足用户需求所需的时间。
- ❑ 确保服务器性能。XenServer 可以优化服务器工作负载的位置, 提高性能和利用率,



同时改进资源池内的服务器准备情况。这样便可确保始终能够达到应用的要求和预期的性能标准，帮助企业加快向生产环境中交付新应用的速度。

- 最大限度地减少服务器宕机。XenServer 可以有效地减少计划内服务器宕机，减小故障的影响，预防灾难并搭建始终可用的虚拟基础架构。服务器和应用的升级可以在正常的工作时间内完成。这样就可以减小对用户生产率的影响，节约成本，使 IT 人员在晚上和周末都能正常休息。

#### 4.4.2 Citrix Xenserver 部署

Citrix Xenserver 与传统虚拟机类软件不同，它无需底层原生操作系统的支持，也就是说 XenServer 本身就具备了操作系统的功能，是能直接安装在服务器上引导启动并运行的，其稳定性较 Hyper-V 高，对 Windows 2008 R2 及 Linux Server 提供了良好的支持，此外，Citrix 还提供了 XenCenter 工具，可通过图形化的控制界面，直观地管理和监控 XenServer 服务器的工作。例如：将一台性能强劲的服务器划分成多台服务器，让这些服务器同时运行以提供各种应用服务，节省硬件投资，也方便管理。假设公司只有一台 Web 服务器，因为公司业务发展，现在需要增加邮件服务、BBS 客户服务，实际上无须购买三台物理服务器来分别实现上述功能，而只须使用 XenServer 在一台物理服务器上创建三台虚拟的服务器，运行各自的操作系统和应用服务，某台虚拟服务器的宕机也不会影响到其他虚拟服务器。

XenServer 的部署很简单，安装界面人性化且有详细的提示，前提是：安装 XenServer 的 PC 服务器没有安装任何操作系统，且拥有至少一块网卡。下载 XenServer 的安装 ISO 文件，刻录成光盘插入服务器的光驱后，启动服务器即可安装。



**提示** XenServer 官网地址为：<http://www.citrix.com/products/xenserver/>

XenServer 下载地址为：<http://downloadns.citrix.com.edgesuite.net/7281/XenServer-6.2.0-install-cd.iso>

XenServer 官方安装文档可以从如下网址下载：[http://www.citrix.com/content/dam/citrix/en\\_us/documents/products-solutions/citrix-xenserver-quick-installation-and-licensing-guide.pdf](http://www.citrix.com/content/dam/citrix/en_us/documents/products-solutions/citrix-xenserver-quick-installation-and-licensing-guide.pdf)

#### 4.4.3 基于 XenCenter 的虚拟服务器管理

XenServer 可通过客户机安装管理工具 XenCenter 进行管理。XenCenter 采用基于图形用户界面的管理控制台，该控制台可安装在任何 Windows PC 或服务器上。XenServer 安装完毕后，直接使用浏览器访问 XenServer 的 IP 地址，下载 XenCenter 并安装。也可在 XenServer 的安装光盘或 ISO 文件内找到 XenCenter 的安装程序，双击执行文件 XenCenter.msi 后，按提示一步步安装即可，如图 4-3 和图 4-4 所示。



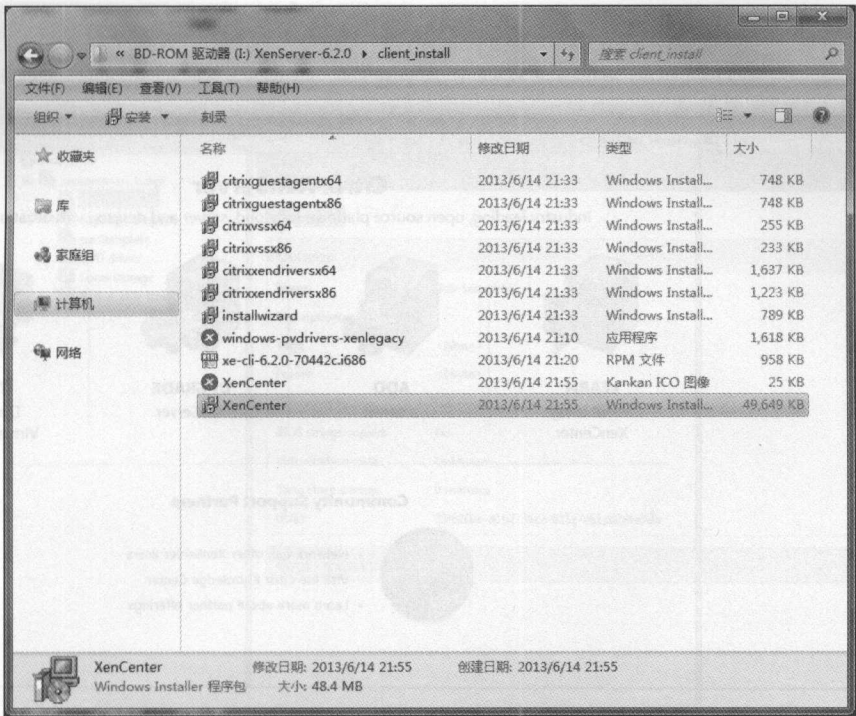


图 4-3 XenCenter 安装包

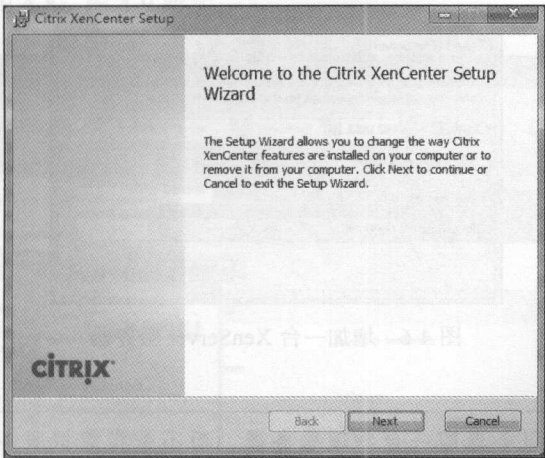


图 4-4 XenCenter 安装程序

安装完 XenCenter 后启动，其界面如图 4-5 所示。

在 XenCenter 处点击鼠标右键，选择 Add 或点击上方工具条中的 Add New Server 按钮增加一台 XenServer 服务器，输入服务器 IP 和 root 账户密码。如图 4-6 所示。

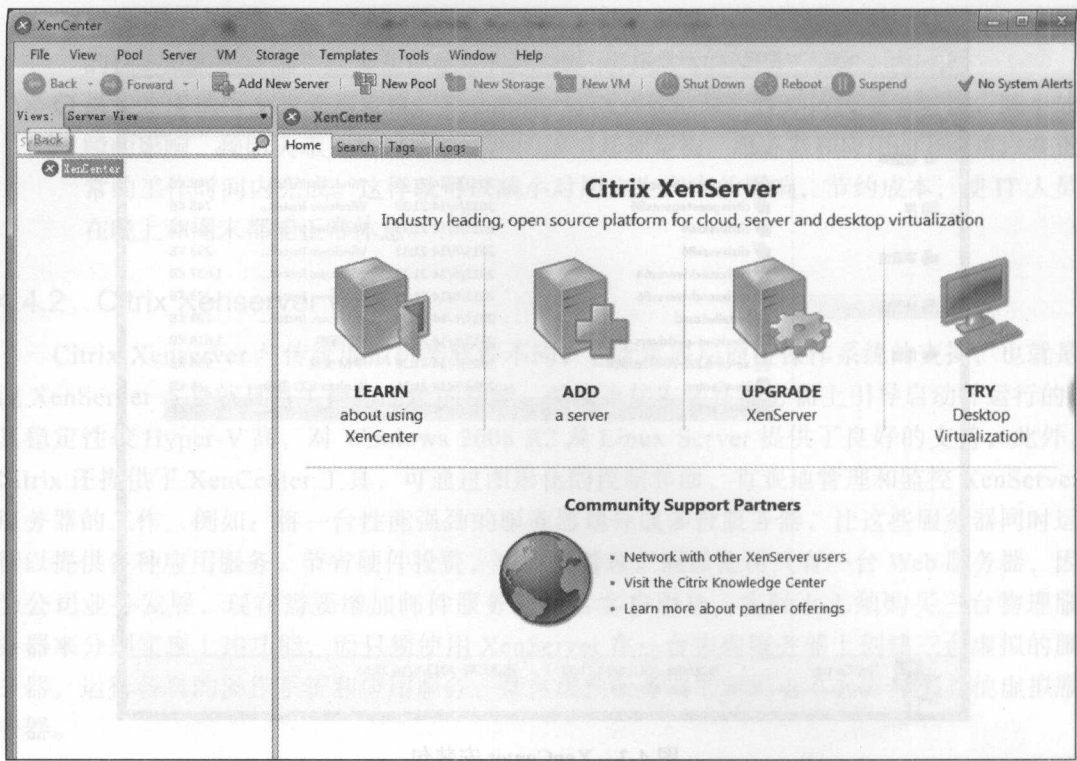


图 4-5 XenCenter 界面

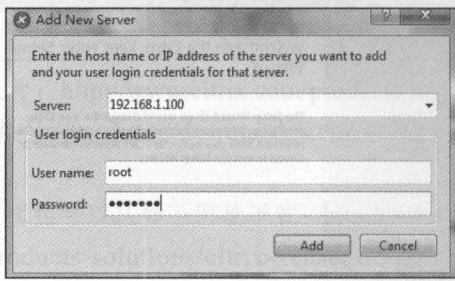


图 4-6 增加一台 XenServer 服务器

**提示** 一台 XenServer 服务器即一台物理服务器，而不是指虚拟服务器，XenCenter 可同时管理多台 XenServer 服务器及运行在 XenServer 服务器上的虚拟服务器。

当 XenCenter 成功连接服务器后，可看到服务器的信息，如图 4-7 所示，在此 XenServer 服务器上一共建立了三台虚拟服务器，服务器前面的小图标表示服务器当前的状态，其中 ddb-template 服务器正在运行中，另外两台服务器处于关机状态。

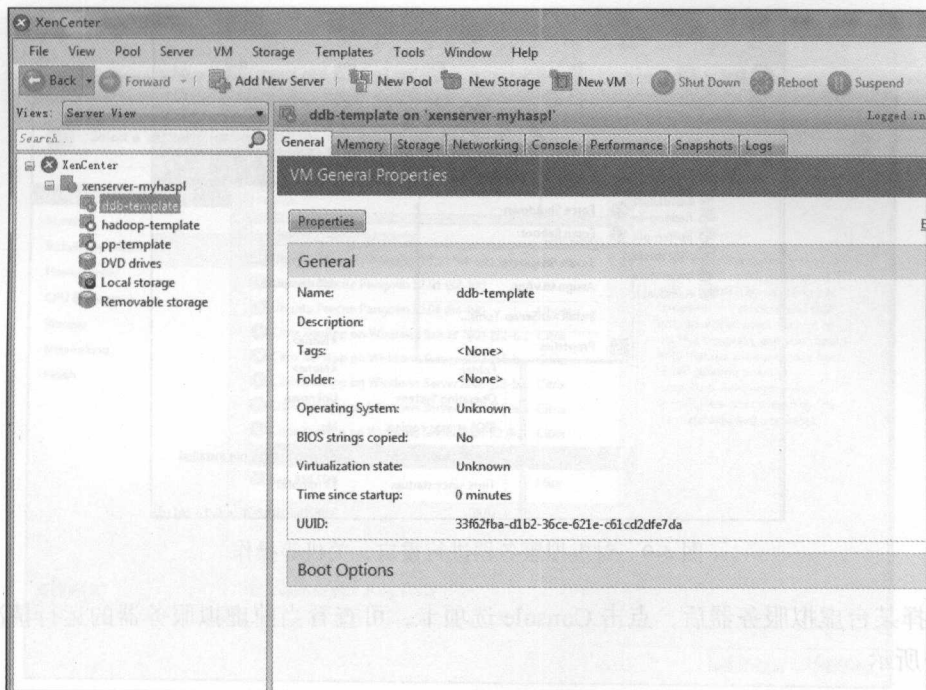


图 4-7 服务器的信息

选择某台虚拟服务器，点击鼠标右键，将出现管理菜单，可对虚拟服务器进行管理，比如重启、关机等，如图 4-8、图 4-9 所示。

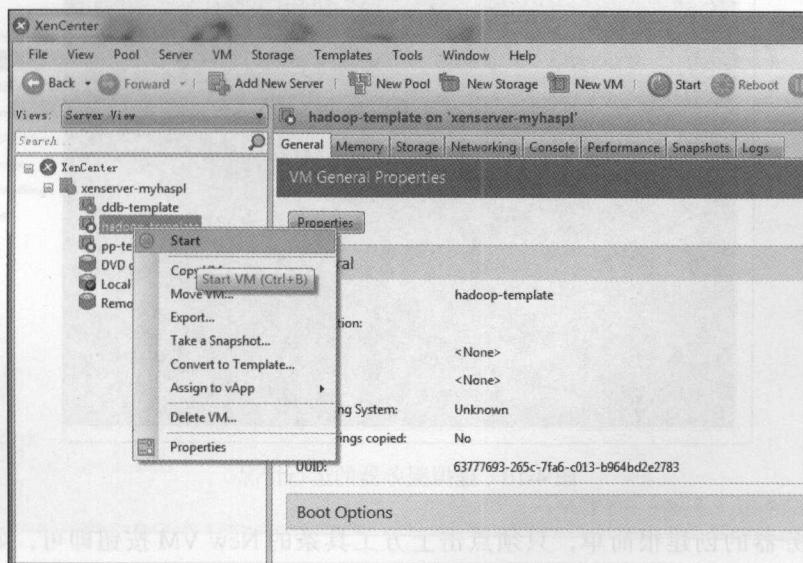


图 4-8 对虚拟服务器进行启动、拷贝、创建快照等操作

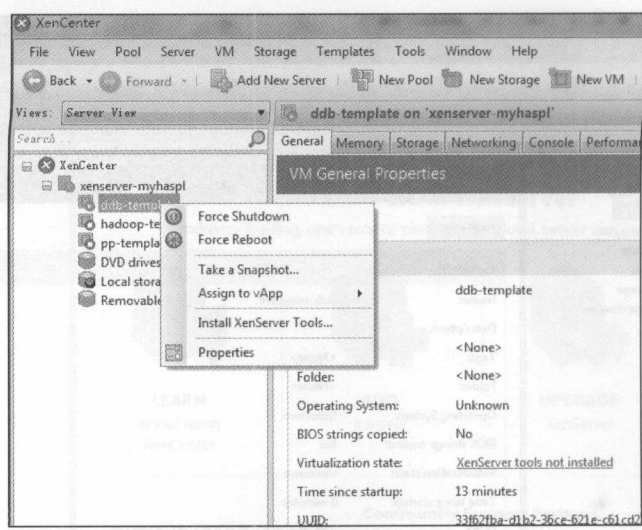


图 4-9 对虚拟服务器进行重启、关机等操作

选择某台虚拟服务器后，点击 Console 选项卡，可查看当前虚拟服务器的运行情况，如图 4-10 所示。

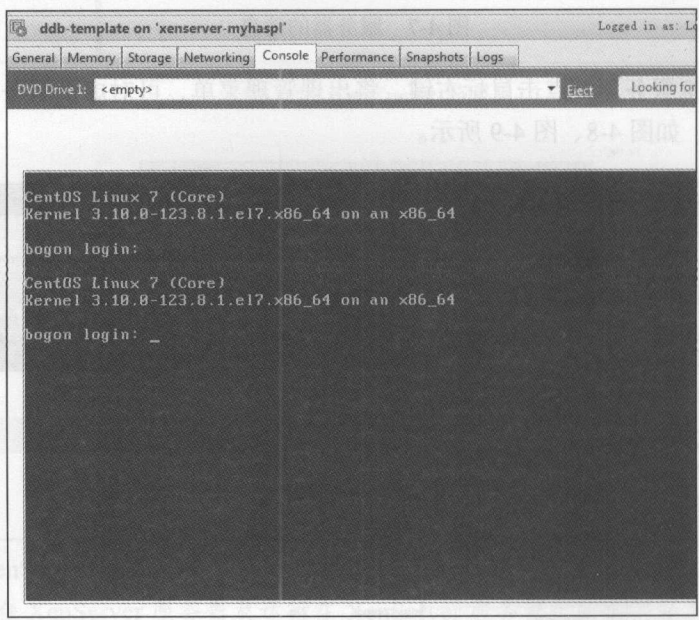


图 4-10 虚拟服务器的运行情况

虚拟服务器的创建很简单，只须点击上方工具条的 New VM 按钮即可，如图 4-11 至图 4-19 所示。创建完虚拟服务器后，将操作系统的安装光盘放入 XenServer 服务器的光驱，



即可继续在该虚拟服务器上安装操作系统。

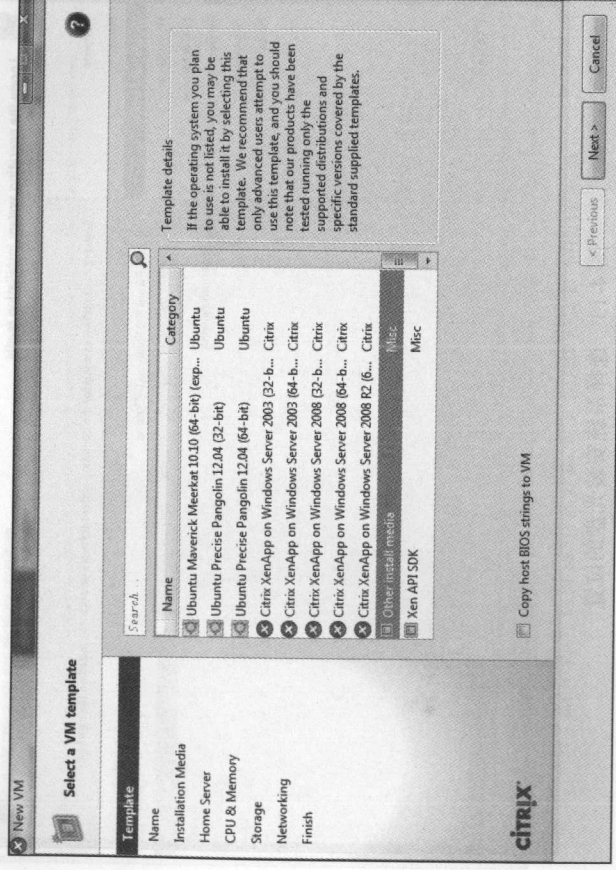


图 4-11 选择虚拟机模板，可不选择已有模板

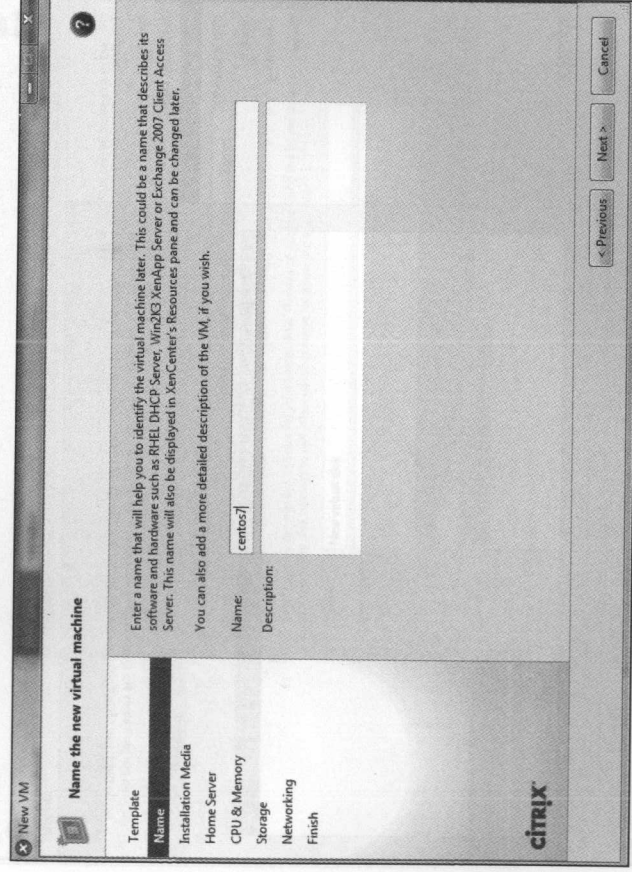


图 4-12 虚拟服务器命名



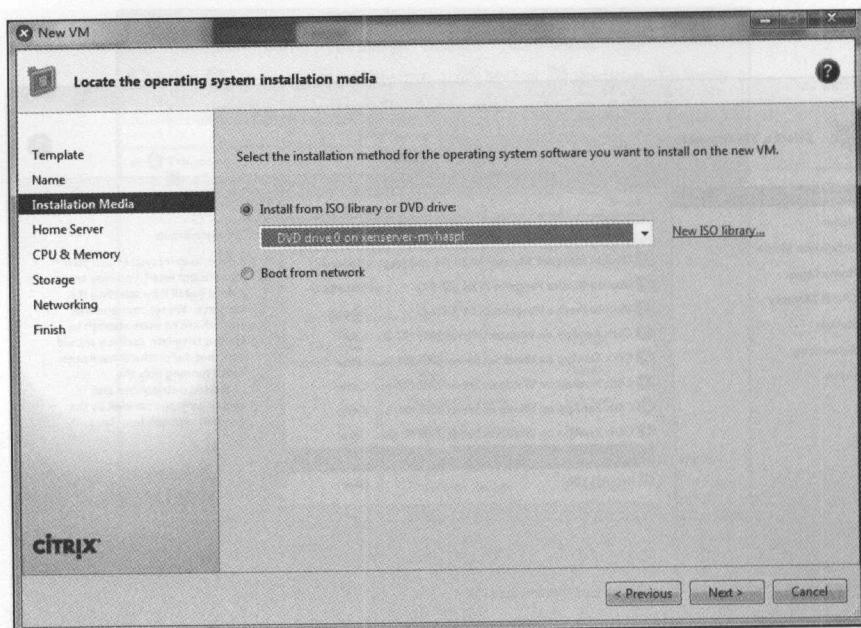


图 4-13 选择系统安装光盘的位置

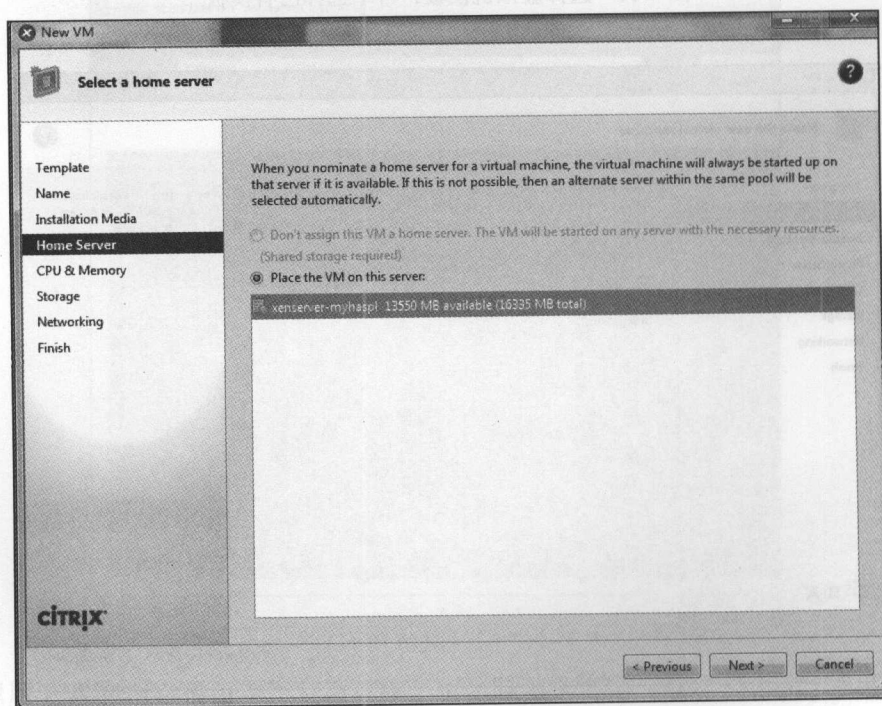


图 4-14 选择虚拟服务器空间占有的物理硬盘

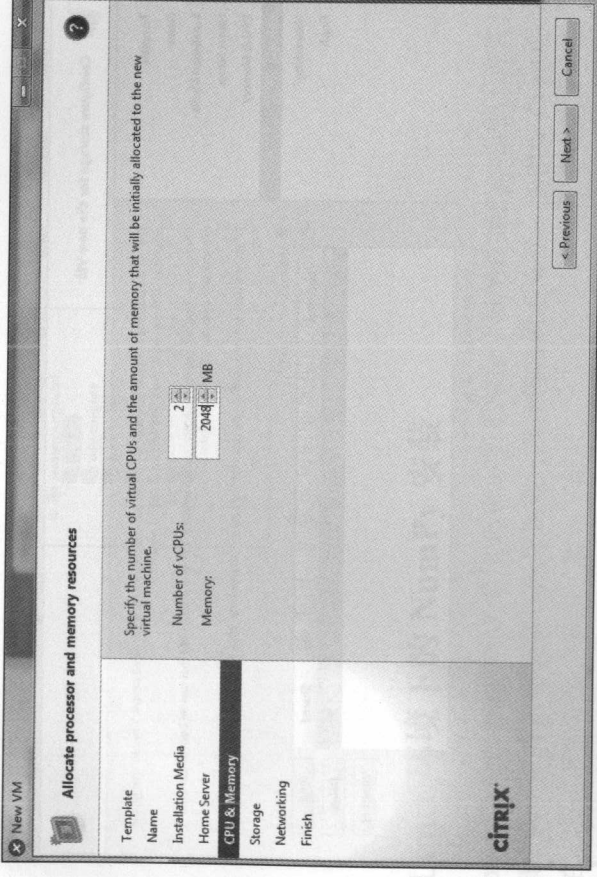


图 4-15 选择虚拟服务器需要使用的 CPU 和内存数量

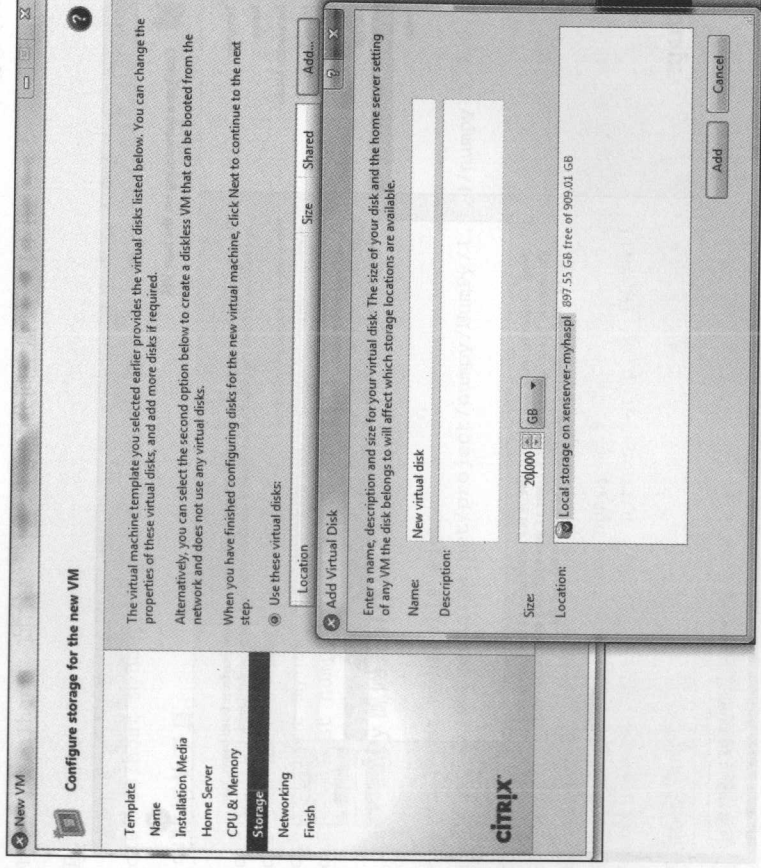


图 4-16 为虚拟服务器分配硬盘空间

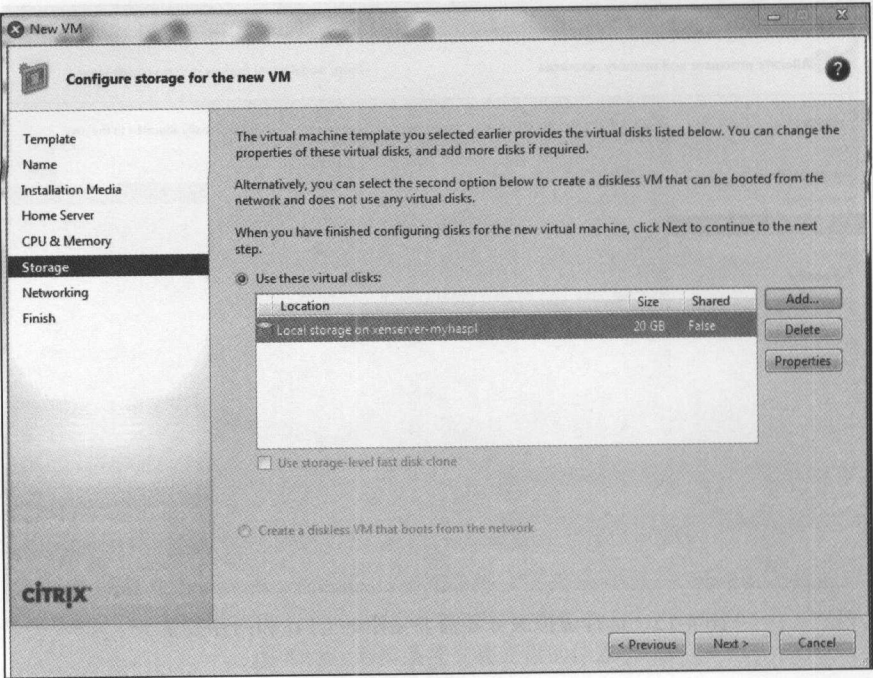


图 4-17 硬盘空间分配完毕，可继续分配

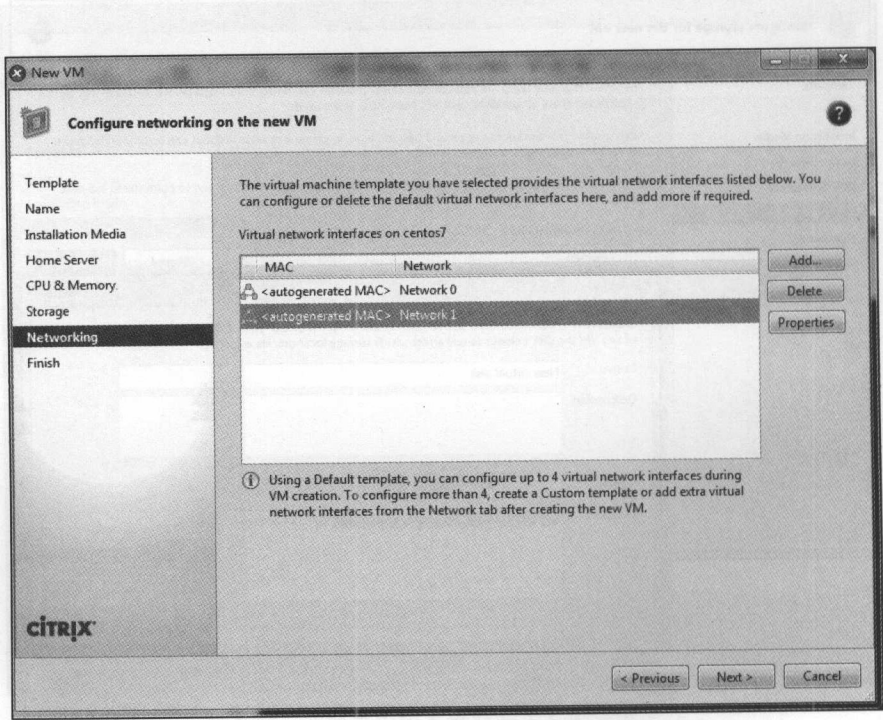


图 4-18 选择虚拟服务器使用的物理网卡

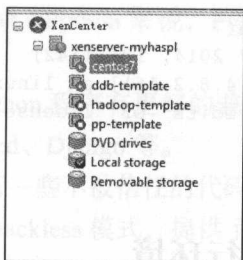


图 4-19 创建成功后的虚拟服务器 centos7

## 4.5 Linux 环境下的 NumPy 安装

NumPy (Numeric Python) 是用 Python 实现的一个科学计算包，提供了许多高级的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库。下面以 CentOS 为例，讲解 NumPy 在 Linux 下的安装，安装过程如下所示。

1) 下载 NumPy，网址为：<http://www.scipy.org/scipylib/download.html>。

2) 系统更新：

```
[myhaspl@localhost ~]$ su
```

密码：

```
[root@localhost myhaspl]# yum install update
```

3) 安装相关工具：

```
[root@localhost myhaspl]# yum install wget
```

```
[root@localhost myhaspl]# yum install unzip
```

```
[root@localhost myhaspl]# yum install gcc
```

```
[root@localhost numpy-1.9.0]# yum install python-devel
```

4) 下载 NumPy 源码并解压：

```
[root@localhost myhaspl]# wget
```

```
http://jaist.dl.sourceforge.net/project/numpy/NumPy/1.9.0/numpy-1.9.0.zip
```

5) 安装 NumPy：

```
[root@localhost myhaspl]# unzip numpy-1.9.0.zip
```

```
[root@localhost myhaspl]# cd numpy-1.9.0
```

```
[root@localhost numpy-1.9.0]# python setup.py install
```

6) 安装完毕后重启：

```
[root@localhost numpy-1.9.0]# reboot
```

7) 测试安装是否成功，如果能导入 NumPy 库，则表示安装成功：



```
[myhaspl@localhost ~]$ python
Python 2.7.5 (default, Jun 17 2014, 18:11:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy as np
>>>
```

## 4.6 Linux 环境下的 R 运行环境

R 语言是一套完整的数据处理、计算和制图软件系统，主要用于统计分析。以 CentOS 为例，可依次输入以下命令，安装 R 运行环境：

```
# yum install gcc-gfortran
# yum install gcc gcc-c++
# yum install readline-devel
# yum install libXt-devel
# wget http://mirror.bjtu.edu.cn/cran/src/base/R-3/R-3.1.1.tar.gz
# tar zxvf R-3.1.1.tar.gz
# cd R-3.1.1
# ./configure
# make
# make install
# make check

# ln -s /home/myhaspl/R-3.1.1/bin/R /usr/local/bin/R

$R
```

## 4.7 PyPy 编译器

### 4.7.1 PyPy 概述

CPython 是用 C 语言实现的 Python 解释器，也是官方的并且是最广泛使用的 Python 解释器。但它并不是唯一的选择，它使用字节码的解释器，任何的程序源代码在执行之前都要先编译成字节码，这样必然会影响到 Python 程序的执行效率。此外，CPython 还存在一个问题，在多处理器的计算机上使用 CPython 要受 GIL (Global Interpreter Lock) 的约束，GIL 使得 CPython 不能进行并发编程，要做到并发编程，就必须为每一个线程运行一个解释器，但是它们之间的通信就会非常困难。

PyPy 是 Python 开发者为了更好地 Hack Python 而创建的项目，它比 CPython 更加灵活、更容易实施。PyPy 支持 Python 语言的所有核心部分及大多数的 Python 语言标准库函数模块，并且通过了 Python 语言的 test suite。更为重要的是 PyPy 实现了动态编译，提供了 JIT 编译器和沙盒功能，运行速度比 CPython 要快很多，还可以安全地运行一些不被信任的代码，此外，PyPy 还有支持微线程的版本。目前 PyPy 的最新版主要拥有以下特征：

□ 更快的速度。在 JIT(Just-in-Time) 编译器的帮助下，Python 程序能在 PyPy 下跑得更快。



- ❑ 更高的内存使用效率。相对 CPython 来说, PyPy 能更有效地利用内存, 尤其是几百兆或更多的内存。
- ❑ 兼容性。PyPy 对现有的 Python 程序有更好的兼容性, 它不仅支持 cffi, 还能运行流行的 Python 库, 比如 Twisted、Django 等。
- ❑ 沙盒。PyPy 可以安全地运行一些不被信任的代码。
- ❑ Stackless。PyPy 原生支持 Stackless 模式, 提供了真正的多线程并发。



关于 PyPy 的更多资料可以在官网 <http://www.pypy.org/> 中查看。

## 4.7.2 PyPy 安装与配置

下面以 64 位 centos7 (最小化安装) 为例, 讲解如何安装和配置 PyPy。

首先, 依次输入以下命令, 下载并编译 PyPy:

```
# yum install gcc
# yum install wget
# wget https://bitbucket.org/pypy/pypy/downloads/pypy-2.3.1-src.tar.bz2
# tar jxvf pypy-2.3.1-src.tar.bz2
# yum install libffi-devel
# yum install expat-devel
# yum install ncurses-devel
# yum install bzip2-devel
# yum install openssl-devel
[root@localhost myhaspl]# cd pypy-2.3.1-src/pypy/goal
# python ../../rpython/bin/rpython -Ojit targetpypystandalone
```

然后, 安装适用于 PyPy 的 numpy 包:

```
# git clone https://bitbucket.org/pypy/numpy.git
# cd numpy
# pypy setup.py install
```

最后, 为方便 PyPy 的调用, 可以使用 ln 命令建立软链接, ln 是 Linux 中一个非常重要的命令, 它的功能是为某一个文件在另外一个位置建立一个不同的链接, 具体用法如下:

```
ln -s 源文件 目标文件
```

## 4.7.3 PyPy 性能

PyPy 执行 Python 程序的速度比 CPython 要快很多。下面以除法与累加混合计算为例, 验证 PyPy 的效率。测试用 Python 程序如下:

```
import time
start=time.clock()
sum=0
i=1.0
```

```
while (i<100000000):
    sum+=i/2.22
    i=i+1
print "sum:%f"%sum
end = time.clock()
print "seconds:%f"%( end- start)
```

分别用 CPython 和 PyPy 运行上面的程序，结果如下：

```
$ python pythontest.py
sum:22522520270270.273438
seconds:4.090000
$ ./pypy pythontest.py
sum:22522520270270.273438
seconds:0.256000
```

从上面的程序运行效果来看，PyPy 计算只用了 0.256 000 秒，而 CPython 则花了 4.090 000 秒，PyPy 对运行效率的提升是数量级的，也是显而易见的。

#### 4.7.4 PyPy 实践之 Lempel-Ziv 压缩

Lempel-Ziv 算法采用动态无损数据压缩算法对字符串进行动态编码和解码，以达到压缩和解压文本的目的。Lempel-Ziv 算法基于在正文流中很可能会出现词汇和短语重复的情况而开发的，当出现一个重复时，将重复的序列用一个短的编码来代替，压缩程序扫描这样的重复，同时生成编码来代替重复序列，随着时间的推移，编码可以重用来捕获新的序列，此外，Lempel-Ziv 算法设计成解压程序能够以编码和原始数据序列推导出当前的映射。

Lempel-Ziv 算法的关键在于查找重复的序列，当碰到一个重复时，算法继续扫描直到该重复序列终止为止。以下面这段文本为例对 Lempel-Ziv 编码进行讲解，假设需要压缩的文本如下：

abaaaabaabaaabbbbbbbabbbbbbbabbbbbbaababaababa...

应用 Lempel-Ziv 算法逐步对上述文本进行编码：

- 1) 建立一个码表空字典 DICT。
- 2) 读入第 1 个字符“a”，DICT 中无“a”字符，且为空，因此将“a”增加到 DICT 中，索引为 1，编码为“0a”。最终编码结果为：“0a”。
- 3) 读入第 2 个字符“b”，DICT 中无“b”字符，因此将“b”增加到 DICT 中，索引为 2，编码为“0b”。最终编码结果为：“0a0b”。
- 4) 读入第 3 个字符“a”，DICT 中有“a”字符且索引为 1，此时，将紧跟其后的第 4 个字符“a”与第 3 个字符“a”合并成一个码段“aa”，编码为“1a”，然后将“aa”增加到 DICT 中，索引为 3。最终编码结果为：“0a0b1a”。
- 5) 第 4 步已经处理过第 4 个字符了，因此从第 5 个字符开始，读入第 5 个字符“a”和第 6 个字符“a”，DICT 中有“aa”字符且索引为 3，此时，将紧跟其后的第 7 个字符“b”与第 5、6 个字符组成的串“aa”合并成一个码段“aab”，编码为“3b”，然后将“aab”增加到 DICT 中，索引为 4。最终编码结果为：“0a0b1a3b”。

接下来，按类似的方法继续处理，直到读取完毕所有需要压缩的字符才结束，最终形成表 4-1 所示的编码结果。

表 4-1 Lempel-Ziv 算法编码

编码	0a	0b	1a	3b	4a	4b	2b	7b	1b	8b	5b	11a	...
字典索引	1	2	3	4	5	6	7	8	9	10	11	12	...
码段	a	b	aa	aab	aaba	aabb	bb	bbb	ab	bbbb	aabab	aababa	...

下面用 Python 来实现 Lempel-Ziv 压缩算法，程序如下所示。

```
# -*- coding: utf-8 -*-
#lempel-ziv 算法
#code:myhaspl@myhaspl.com
#4-1.py
# 待压缩字符串
my_str="Ubuntu 14.04 LTS includes a wealth of smart filters to make it faster
and easier to find the content you need, whether it's stored on your computer or on
the web.Type any query into the Dash home and the Smart Scopes server will determine
which categories of content are the most relevant to your search, returning only
the best results. The server constantly improves its results by learning which
categories and results are most useful to you over time."

# 码表
codeword_dictionary={}
# 待压缩文本长度
str_len=len(my_str)
# 码字最大长度
dict_maxlen=1
# 将解析文本段的位置（下一次解析文本的起点）
now_index=0
# 码表的最大索引
max_index=0

#
compressed_str=""

while (now_index<str_len):
    # 向后移动步长
    mystep=0
    # 当前匹配长度
    now_len=dict_maxlen
    if now_len>str_len-now_index:
        now_len=str_len-now_index
    # 查找到的码表索引，0 表示没有找到
    cw_addr=0
    while (now_len>0):
        cw_index=codeword_dictionary.get(my_str[now_index:now_index+now_len])
        if cw_index!=None:
            # 找到码字
            cw_addr=cw_index
            mystep=now_len
```

```

        break
    now_len-=1
    if cw_addr==0:
        # 没有找到码字, 增加新的码字
        max_index+=1
        mystep=1
        codeword_dictionary[my_str[now_index:now_index+mystep]]=max_index
        print "don't find the Code word,add Code word:%s index:%d"%(my_str[now_
index:now_index+mystep],max_index)
    else:
        # 找到码字, 增加新的码字
        max_index+=1
        codeword_dictionary[my_str[now_index:now_index+mystep+1]]=max_index
        if mystep+1>dict_maxlen:
            dict_maxlen=mystep+1
            print "find the Code word:%s add Code word:%s index:%d"%(my_str[now_
index:now_index+now_len],my_str[now_index:now_index+mystep+1],max_index)

# 计算压缩后的结果
cwdindex='%d'%cw_addr
if cw_addr==0:
    cwlater=my_str[now_index:now_index+1]
    now_index+=1
else:
    now_index+=mystep
    cwlater=my_str[now_index:now_index+1]
    now_index+=1
cw=cwdindex+cwlater
compressed_str+=cw
print "\n-----\n"
print my_str
print "\n*****\n"
print compressed_str
print "\n-----\n"

```

现在用以上程序将下面的文本进行压缩:

Ubuntu 14.04 LTS includes a wealth of smart filters to make it faster and easier to find the content you need, whether it's stored on your computer or on the web. Type any query into the Dash home and the Smart Scopes server will determine which categories of content are the most relevant to your search, returning only the best results. The server constantly improves its results by learning which categories and results are most useful to you over time.

压缩结果为:

```

00U0b0u0n0t3 01040.008 0L0T0S0 0i4c0l3d0e0s15a15w20a18t0h15o0f15s0m0a0r5
28i25e32s15t0o15m31k20 16t15f31s5e32 31n0d15e44i20r37o43i4d37h41c38n45n33y38u15n20e48,
23h20t26e46i5e01 21t38r20d27n15y60r15c38m0p3t51 73 57 5h41w20b9T0y80e22n89
0q3e32y15i4t38 85e15D44h15h79e91d55e15S30a32t106c38p20s29e32v82w16l18 48e45r30i4e64i0c26
122a45g73i111 38f78o97e97 31r41t66 30o72 32e18e0v47t52 89o3r112a32c26,15r65u32n16n0g7519
2t134b111t147e2lu25s9 13h41s51v82c57s5a97l92i30p32o139e71i5s156s31l72 2y15l24r4i4g121c12
3c31t20g126e71a54 137s174t186r41m38s33u21e28u116t98y60 38v82t16m20.

```

下面使用 PyPy 运行程序 4-1.py, 将程序依次读入文本, 扩充码表字典, 最终输出压缩结果, 如下:

```
$pypy 4-1.py
don't find the Code word,add Code word:U index:1
don't find the Code word,add Code word:b index:2
don't find the Code word,add Code word:u index:3
don't find the Code word,add Code word:n index:4
don't find the Code word,add Code word:t index:5
find the Code word:u add Code word:u index:6
don't find the Code word,add Code word:l index:7
don't find the Code word,add Code word:4 index:8
don't find the Code word,add Code word:. index:9
...
...
137s174t186r41m38s33u21e28u116t98y60 38v82t16m20.
```

程序 4-1.py 在压缩时生成了码表, 这就带来了一个问题: 码表字典文件会很大, 而每个压缩文件都要将码表附上, 这样就无法达到压缩文件尺寸的作用。实际上 Lempel-Ziv 算法在压缩文件中并不需要存放码表, 可在解压时自动生成码表, 此外, 同时还可根据被压缩文件的大小确定索引是整型、长整型还是字节型, 这样就可动态地调整在压缩文件中索引所占有的空间, 保证了压缩率。下面用 Python 实现对文本文件进行 Lempel-Ziv 压缩与解压缩, 程序如下所示:

```
# -*- coding: utf-8 -*-
#lempel-ziv 压缩与解压缩文件
#code:myhaspl@myhaspl.com
#4-2.py
import struct
import sys
txtfilename=sys.argv[1]
compressfn=sys.argv[2]
mystr=""
print "\n读取源文件 ".decode("utf8")
mytextfile= open(txtfilename,'r')
try:
    mystr=mytextfile.read()
finally:
    mytextfile.close()
my_str=mystr
# 码表
codeword_dictionary={}
# 待压缩文本长度
str_len=len(my_str)
# 码字最大长度
dict_maxlen=1
# 将解析文本段的位置 (下一次解析文本的起点)
now_index=0
# 码表的最大索引
max_index=0
```



```

# 压缩后的数据
print "\n生成压缩数据中".decode("utf8")
compresseddata=[]
while (now_index<str_len):
    # 向后移动步长
    mystep=0
    # 当前匹配长度
    now_len=dict_maxlen
    if now_len>str_len-now_index:
        now_len=str_len-now_index
    # 查找到的码表索引, 0 表示没有找到
    cw_addr=0
    while (now_len>0):
        cw_index=codeword_dictionary.get(my_str[now_index:now_index+now_len])
        if cw_index!=None:
            # 找到码字
            cw_addr=cw_index
            mystep=now_len
            break
        now_len-=1
    if cw_addr==0:
        # 没有找到码字, 增加新的码字
        max_index+=1
        mystep=1
        codeword_dictionary[my_str[now_index:now_index+mystep]]=max_index
        print "don't find the Code word,add Code word:%s index:%d"%(my_str[now_
index:now_index+mystep],max_index)
    else:
        # 找到码字, 增加新的码字
        if now_index+mystep+1<str_len:
            # 如果文件尾部正好是字典中的码字, 后面再无内容时, 这部分将不会被执行
            max_index+=1
            codeword_dictionary[my_str[now_index:now_index+mystep+1]]=max_index
            if mystep+1>dict_maxlen:
                dict_maxlen=mystep+1
            print "find the Code word:%s add Code word:%s index:%d"%(my_
str[now_index:now_index+now_len],my_str[now_index:now_index+mystep+1],max_index)
        # 压缩后的结果
        cwdindex=cw_addr
        if cwdindex==0:
            cwlater=my_str[now_index:now_index+1]
            now_index+=1
        else:
            now_index+=mystep
            if now_index>=str_len:
                # 文件已经读取完毕, 后面无内容
                cwlater=""
            else:
                cwlater=my_str[now_index:now_index+1]
                now_index+=1
        cw=(cwdindex,cwlater)
        compresseddata.append(cw)

```

```
print "\n生成压缩数据头部 ".decode("utf8")
# 生成数据压缩大小格式, 65535 为 H 能容纳的最大数据
if len(codeword_dictionary)<=65535:
```

```
    FLAG_FMT='H'
    FLAG_LEN='2'
```

```
else:
```

```
    FLAG_FMT='I'
    FLAG_LEN='4'
```

```
# 压缩数据写入文件
```

```
print "\n将压缩数据写入压缩文件中 ".decode("utf8"),
```

```
lzv_file = open(compressfn, 'wb')
```

```
try:
```

```
    # 写入压缩数据的头部信息
```

```
    # 压缩数据长度
```

```
    lzv_len=len(compresseddata)
```

```
    lzv_file.write(struct.pack('I', lzv_len))
```

```
    # 数据解码大小格式:
```

```
    lzv_file.write(struct.pack('!s!s', FLAG_FMT, FLAG_LEN))
```

```
    # 写入压缩数据
```

```
    for temp_data in compresseddata:
```

```
        temp_key, temp_later=temp_data
```

```
        temp_wdata=struct.pack(FLAG_FMT+'!s!', temp_key, temp_later)
```

```
        lzv_file.write(temp_wdata)
```

```
        print " .",
```

```
finally:
```

```
    lzv_file.close()
```

```
# 解压缩数据
```

```
print "\n读入压缩数据中 ".decode("utf8"),
```

```
my_compresseddata=[]
```

```
my_codeword_dictionary={}
```

```
lzv_file = open(compressfn, 'rb')
```

```
try:
```

```
    # 读入压缩数据的长度
```

```
    lzv_len,=struct.unpack("I", lzv_file.read(4))
```

```
    # 读入数据解码大小格式
```

```
    MY_FLAG_FMT, MY_FLAG_LEN=struct.unpack('!s!s', lzv_file.read(2))
```

```
    # 读入压缩数据
```

```
    tmp_count=1
```

```
    while tmp_count<=lzv_len:
```

```
        # 读入压缩数据
```

```
        temp_data =lzv_file.read(int(MY_FLAG_LEN)+1)
```

```
        temp_key, temp_later=struct.unpack(MY_FLAG_FMT+'!s!', temp_data)
```

```
        my_compresseddata.append((temp_key, temp_later))
```

```
        tmp_count+=1
```

```
        print " .",
```

```
    print "\n读入压缩数据成功 ".decode("utf8")
```

```
finally:
```

```
    lzv_file.close()
```

```
print "\n解压中 ".decode("utf8"),
```

```
# 解压缩
```

```

uncompress_str=''
# 解码后的数据
uncompressdata=[]
my_maxindex=0
# 解码,同时重构码表
for (cwkey,cwlaster) in my_compresseddata:
    if cwkey==0:
        my_maxindex+=1
        my_codeword_dictionary[my_maxindex]=cwlaster
        uncompressdata.append(cwlaster)
    else:
        my_maxindex+=1
        my_codeword_dictionary[my_maxindex]=my_codeword_dictionary[cwkey]+cwlaster
        uncompressdata.append(my_codeword_dictionary[cwkey])
if cwlaster!='\0':
    uncompressdata.append(cwlaster)
print ".",
uncompress_str=uncompress_str.join(uncompressdata)
uncompressstr=uncompress_str
print "\n将解压结果写入文件中..\n".decode("utf8")
uncompress_file= open('uncompress.txt','w')
try:
    uncompress_file.write(uncompressstr)
    print "\n解压成功,已解压到 uncompress.txt! \n".decode("utf8")
finally:
    uncompress_file.close()

```

下面以图 4-20 所示的文本为例进行压缩和解压缩操作。

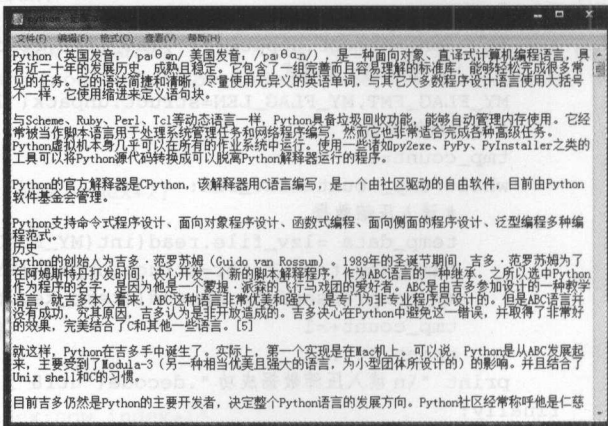
首先,调用 PyPy 运行 4-2.py,该程序先进行压缩形成压缩文件,再打开压缩文件解压将文件还原为 uncompress.txt。运行结果如下:

```
$ pypy 4-2.py python.txt python.lzv
```

```

.....
find the Code word: C    add
Code word: CP index:9938
    index:9939de word:ython    add
Code word:ython
find the Code word:
^    add Code word:
^ h index:9940
find the Code word:ttp    add
Code word:ttp: index:9941
find the Code word://    add
Code word://e index:9942
find the Code word:dit    add
Code word:ditr index:9943
find the Code word:a.    add
Code word:a.o index:9944
生成压缩数据头部
将压缩数据写入压缩文件中
.....

```



将解压结果写入文件中。  
解压成功，已解压到 uncompress.txt！

然后，查看一下解压缩效果，可见解压缩成功，结果如下所示：

```
$ cat uncompress.txt
```

Python（英国发音：/paɪθən/；美国发音：/paɪθɑːn/）是一种面向对象、直译式计算机编程语言，具有近二十年的发展历史，成熟且稳定。它包含了一组完善而且容易理解的标准库，能够轻松完成很多常见的任务。它的语法简洁和清晰，尽量使用无异义的英语单词，与其他大多数程序设计语言使用大括号不一样，它使用缩进来定义语句块。

最后，运行 ls 命令查看算法压缩效果，结果如下：

```
$ ls -l -h
.....
-rw-rw-r-- 1 deep deep 5.0K Jul  1 20:55 5-2.py
-rw-rw-r-- 1 deep deep  30K Jul  1 20:55 python.lzv
-rw-rw-r-- 1 deep deep  36K Jul  1 20:57 python.txt
-rw-rw-r-- 1 deep deep  36K Jul  1 20:55 uncompress.txt
```

从上面显示的结果可以看到，未压缩前为 36KB，压缩后为 30KB，压缩效果不太明显，Lempel-Ziv 算法对于大的文件压缩效果更好，以 sqlite 3.8.5 的全部源码为例，对它进行压缩，调用 PyPy 再次运行 4-2.py 程序：

```
$ pypy 4-2.py sqlitesrc.txt sqlitesrc.lzv
```

程序运行完毕后，查看压缩效果，结果如下：

```
$ ls -l -h
.....
-rw-rw-r-- 1 deep deep 3.2M Jul  1 21:18 sqlitesrc.lzv
-rw-rw-r-- 1 deep deep 5.2M Jul  1 21:16 sqlitesrc.txt
-rw-rw-r-- 1 deep deep 5.2M Jul  1 21:18 uncompress.txt
没压缩前为 5.2M，压缩后为 3.2M，压缩效果较明显。
```

## 4.8 小结

目前，机器学习主要依靠算法来实现，而算法依靠程序来实现，程序在调试完毕后，就需要投入到实际运行阶段，当人们创造性地将“生产环境”一词拓展到软件工程领域后，软件运行平台的定义就变得严谨、可靠和安全，它特指软件调试完毕并正式启用后实际运行的环境。软件生产环境中最常用的是 Windows Server 和 Linux/UNIX 两种。本章首先分别介绍了这两种生产环境下的基本命令、Shell 脚本及各自的特性，然后重点介绍了虚拟化平台的搭建与管理，最后讲解了 Linux 环境下 Vim 编辑器的使用及相关软件工具的配置。值得一提的是，本章结尾讲述了 PyPy 编译器，PyPy 实现了动态编译，提供了 JIT 编译器和沙盒功能，运行速度比 CPython 要快很多，还可以安全地运行一些不被信任的代码，对于 Python 程序来

说,是一个很不错的运行平台。

## 思考题

在家中找一台闲置不用的 PC,将硬盘格式化,然后下载 Citrix Xenserver 免费版和 Windows Server 2008 R2 免费试用 180 天版,将本章内容亲自实践一次,按以下步骤进行操作:

- (1) 安装和配置好 Citrix Xenserver 与 XenCenter。
- (2) 建立至少两台虚拟服务器,一台用于安装 Windows Server 2008 R2,一台用于安装 Citrix Xenserver。
- (3) 在其中一台虚拟服务器中安装 CentOS 系统。
- (4) 在 CentOS 系统中安装好 R、PyPy、NumPy 等工具软件包,安装完毕后,测试一下各软件包是否能正常运行相关的程序。
- (5) 在 CentOS 系统中编写各种 Linux 命令和 Shell 脚本进行测试和学习。
- (6) 在 CentOS 系统中打开 Vim 编辑器,编辑以下文件内容,并存盘为 R.txt。

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

- (7) 在另一台虚拟服务器中安装 Windows Server 2008 R2。

- (8) 在 Windows Server 2008 R2 中运行 Windows PowerShell 脚本,并配置好 R、Pypy、Python、NumPy 等工具软件包,并测试各软件包是否能正常运行相关的程序。



随着社会的发展,数据早已渗透到各行各业,成为了社会信息记录的重要载体。为了解决数据的查询问题,在数据库发展的早期,SQL 就有过很广泛的应用,但它仍然活跃在数据库领域。近年来,“大数据”一词被炒得很热,大数据已广泛用于承载数据相关的绝大部分业务,如统计、数据挖掘、社交媒体分析、下一代数据库管理等。然而数据等,各大公司已如何处理并分析海量信息。

只有登上山顶,才能看到真正的风光。既然

### 第三部分 Part 3

### 统计分析实战篇

## 统计分析实战篇

概率是数学概率论的基本概念。设随机事件的样本空间为 $\Omega$ , $\Omega$ 中的每一个事件 $A$ ,都有实函数 $P(A)$ ,满足:

非负性:  $P(A) \geq 0$

规范性:  $P(\Omega) = 1$

可加性: 若 $A_1, A_2, \dots, A_n$ 是两两互斥的事件,则

$$P(A_1 \cup A_2 \cup \dots \cup A_n) = P(A_1) + P(A_2) + \dots + P(A_n)$$

在终极的分析中,一切知识都是历史;在抽象的意义下,一切科学都是数学;在理性的基础上,所有的判断都是统计。

——C.R. 劳

## 统计分析基础

统计学用于研究对象的数量性、总体性、变异性,具体说来,就是通过各种统计指标和指标体系来反映对象总体的规模、水平、速度、比例、效益、差异和趋势等。海量数据分析需要一个科学的理论基础,统计学是理论基础之一,统计分析方法也是主要的数据分析方法。

机器学习涉及许多统计分析理论。机器学习应用的统计分析强调实际应用效果,检测损失函数即描述预测与实际之间的偏差;数据分析领域也以统计学为基础,统计学善于对数据建立模型,并对模型做出假设。由此可见,无论是在机器学习方面还是数据分析领域,统计分析理论都有着举足轻重的地位。

本章将以 R 语言为统计分析计算工具讲述统计分析基础,在此之前,请各位读者按照第一部分准备篇中的指导,将 R 语言计算平台搭建好。

### 5.1 数据分析概述

随着数据分析技术的不断发展,数据分析的定义众说纷纭。作者尝试将数据分析定义为:“数据分析是将数据转化成知识,对数据加以详细的研究和概括总结的过程,它用适当的统计方法对收集来的大量数据进行分析,发现数据的价值,挖掘数据的表征的知识。”

在第二次世界大战中,数据分析第一次登上世界舞台。在 Budiansky、Stephen 所著的《Blackett's War》中,讲述了一个由数学家、物理学家组成的团体,通过对数据进行收集和分析,使用数据引导战争的故事。该团队运用数据分析技术对付纳粹潜艇舰队,并把原始雷达系统收集的数据与实际战争结合起来分析,从而使海岸司令部的飞机拥有恰当的飞行路线,实现最高效的监视。同时他们还证明了一个论断:在一个 15 ~ 24 艘船的舰队中,每艘船有 2.3% 被击沉的概率;而在舰队船数高于 45 时,被击沉的概率只有 1.1%。

随着社会的发展,数据库已经深入到各行各业,成为了社会信息记录的重要载体。为了解决数据的查询问题,在数据库发展的早期,SQL语言诞生并迅速发展,目前它仍然活跃在数据库领域。近年来,“大数据”一词被炒得很热,大数据已被用于承载数据相关的绝大部分概念,包括:数据海洋、社交媒体分析、下一代数据管理能力、实时数据等。各大公司已经开始理解并实践探索如何处理并分析大量信息。

无限风光在险峰,只有登上山顶,才能看到真正的风光,既然如此,那我们就开始攀登数据分析这座“大山”吧!

## 5.2 数学基础

统计分析建立在概率与统计的数学基础之上,在此,先简要介绍一下相关数学公式。

### 1. 概率

概率是数学概率论的基本概念。设随机事件的样本空间为 $\Omega$ ,对于 $\Omega$ 中的每一个事件 $A$ ,都有实函数 $P(A)$ ,满足:

非负性:  $P(A) \geq 0$

规范性:  $P(\Omega)=1$

可加性: 对 $n$ 个两两互斥事件 $A_1, \dots, A_n$ 有:  $\sum_{i=1}^n P(A_i) = P\left(\bigcup_{i=1}^n A_i\right)$

任意一个满足上述条件的函数 $P$ 都可以作为样本空间 $\Omega$ 的概率函数,称函数值 $P(A)$ 为 $\Omega$ 中事件 $A$ 的概率。

上面的定义严谨但不易理解,通俗来说,概率是一个0到1之间的实数,是对随机事件发生的可能性的度量。人们有时候会问:“明天会更冷吗?”、“今天的比赛我们会赢得冠军吗?”等,他们是在关注“天气变冷”、“比赛取胜”等事件发生的机会。在数学上,这些事件发生的机会可用一个数来表示,称为概率。概率的主要公式如下。

设事件 $A$ 发生的概率(可能性)为 $P(A)$ ,它不发生的概率(可能性)为 $P(\bar{A})$ ,它们之间的关系为:

$$P(A)=1-P(\bar{A})$$

$n$ 个事件 $A_1, \dots, A_n$ 发生的概率为:

$$P\left(\bigcup_{i=1}^n A_i\right) = \sum_{i=1}^n P(A_i) - \sum_{1 \leq i < j \leq n} P(A_i A_j) + \sum_{1 \leq i < j < k \leq n} (A_i A_j A_k) + \dots + (-1)^{n-1} P(A_1 A_2 \dots A_n)$$

条件概率是事件 $A$ 在另外一个事件 $B$ 已经发生条件下的发生概率。条件概率记为 $P(A|B)$ ,读作“在 $B$ 条件下 $A$ 的概率”。比如,“如果明天更冷,需要多穿一件外套吗?”可用条件概率描述为:假设明天天气更冷为事件 $B$ ,多穿一件外套为事件 $A$ ,在事件 $B$ (天气更冷)发生的情况下,事件 $A$ (多穿外套)发生的概率为多少。下面是条件概率的数学定义。

在同一个样本空间 $\Omega$ 中的事件或者子集 $A$ 与 $B$ ,如果随机从 $\Omega$ 中选出的一个元素属于 $B$ ,那么这个随机选择的元素还属于 $A$ 的概率就定义为在 $B$ 的前提下 $A$ 的条件概率。定义为:

$$P(A|B) = |A \cap B|/|B|$$

再将上式中的分子、分母都除以  $|\Omega|$ , 得到

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

其中,  $P(A \cap B)$  表示联合概率, 指  $A$  和  $B$  两个事件共同发生的概率, 也可以记为  $P(A, B)$  或  $P(AB)$ 。

条件概率在贝叶斯统计中也称为后验概率, 即在考虑和给出相关证据或数据后所得到的条件概率。

现实生活中, 使某事件发生的前提(条件概率中的“条件”)可能不止一个。比如: 在事件  $B_1, \dots, B_n$  发生的前提下, 事件  $A$  发生的概率是多少? 将其定义如下:

$$P(A) = \sum_{i=1}^n P(AB_i) = \sum_{i=1}^n P(B_i) \cdot P(A|B_i)$$

前面几个公式都是根据前提来推测事件发生的概率的, 能不能根据事件发生的概率推测其前提出现的概率呢? 当然可以, 实质上, 这就是根据结论推测原因发生的概率。比如, 为什么前面堵车了(事件  $A$ ), 是因为发生了交通事故(事件  $B_1$ ) 还是路上车辆太多以致拥塞(事件  $B_2$ ), 或是下完大雨道路状况很差(事件  $B_3$ )?

下面的公式定义了事件  $A$  (结论) 发生了, 事件  $B_k$  (原因) 发生的概率,

$$P(B_k|A) = \frac{P(AB_k)}{P(A)} = \frac{P(B_k)P(A|B_k)}{\sum_{i=1}^n P(B_i)P(A|B_i)}$$

## 2. 概率分布

概率分布简称分布, 是数学概率论的一个概念, 广义上指随机变量的概率性质, 狭义上是指随机变量的概率分布函数, 使用最为普遍的是狭义上的定义。在此只讲解狭义概率分布。

1) 随机变量分布。随机变量的实质是函数, 其数学定义为: 设函数  $X$  对于概率空间  $S$  中每一个事件  $e$  都有定义  $X(e)$ , 其中函数值为实数, 同时, 每一个实数  $r$  都有一个事件集合  $A_r$  与其对应, 其中  $A_r = \{e: X(e) \leq r\}$ , 则将  $X$  称作随机变量。

通俗地说, 一个随机试验可能结果(基本事件)的全体组成一个基本空间  $\Omega$ , 随机变量  $X$  是定义在基本空间  $\Omega$  上的取值为实数的函数, 即基本空间  $\Omega$  中每一个点, 也就是每个基本事件都有实轴上的点与之对应。比如: 在一次扔硬币事件中, 将获得的国徽的次数作为随机变量  $X$ , 则  $X$  可以取两个值, 分别是 0 和 1。

如果随机变量  $X$  的取值是有限的或是以下可数无穷尽的值:

$$X = \{x_1, x_2, x_3, \dots\}$$

则称  $X$  为离散随机变量。

如果  $X$  由全部实数或者由一部分区间组成:

$$X = \{x | a \leq x \leq b\}, -\infty < a < b < \infty$$

则称  $X$  为连续随机变量, 连续随机变量的值是不可数并无穷尽的。

设  $P$  为概率测度,  $X$  为随机变量, 则函数  $F(x) = P(X \leq x) (X \in R)$  称为  $X$  的概率分布函数。将随机变量区间的下限定义为  $a$ , 上限定义为  $b$ , 则分布概率  $P(a \leq X \leq b)$  的定义为:



$$p(a \leq X \leq b) = P(X \leq b) - P(X \leq a) \\ = F(b) - F(a)$$

2) 离散随机变量分布。离散型随机变量  $X$  的所有取值与其对应的概率间的关系, 称为离散型随机变量  $X$  的概率分布, 或称为概率函数。可用列表法表示离散随机变量的分布, 如表 5-1 所示。

表 5-1 离散随机变量分布

随机变量 $X$ 值	$x_1$	$x_2$	...	$x_n$	...
随机变量 $X$ 出现的概率 $p$	$p_1$	$p_2$	...	$p_n$	...

表 5-1 中第 1 行是随机变量的值, 第 2 行是每个值出现的概率。比如: 用随机变量描述投掷一枚骰子点数的概率分布, 用随机变量  $X$  表示投掷一枚骰子出现的点数, 概率分布  $p(X=k) = \frac{1}{6} (k=1, 2, \dots, 6)$ , 可用如表 5-2 所示的列表描述。

表 5-2 骰子点数的概率分布

骰子点数 $X$	1	2	3	4	5	6
骰子点数 $X$ 出现的概率 $p$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

离散随机变量的概率分布具有以下两个性质:

$$p_k \geq 0 \quad (k=1, 2, \dots, n)$$

$$\sum_{k=1}^n p_k = 1$$

3) 0-1 分布。伯努利试验是只有两种可能结果的单次随机试验, 即随机变量  $X$  只有两个值 0 或 1, 0-1 分布又名伯努利分布或者两点分布, 是一个离散型概率分布。若伯努利试验成功, 则随机变量取值为 1, 否则随机变量取值为 0, 成功概率为  $p (0 \leq p \leq 1)$ , 失败概率为  $q=1-p$ 。0-1 分布的概率质量函数 (概率质量函数的值就是离散随机变量分布的概率) 定义如下:

$$P(X=k) = p^k (1-p)^{1-k}, k=0, 1$$

4) 二项分布。二项分布即重复  $n$  次独立的伯努利试验, 在每次试验中只有两种可能的结果, 而且两种结果发生与否互相对立, 并且相互独立, 与其他各次试验结果无关。每次试验的成功概率为  $p$ , 当  $n=1$  时, 二项分布就是 0-1 分布。

如果随机变量  $X$  服从参数为  $n$  和  $p$  的二项分布, 可记为  $X \sim B(n, p)$ ,  $n$  次试验正好得到  $k$  次成功的概率为下面的概率质量函数:

$$P(X=k) = C_n^k p^k (1-p)^{n-k}, k=0, 1, \dots, n$$

$$\text{其中, } C_n^k = \frac{n!}{k!(n-k)!}。$$

5) Poisson 分布。Poisson 分布又名泊松分布, 主要描述单位时间内随机事件发生的次数的概率分布。比如: 某服务设施在一定时间内收到的服务请求的次数、电话交换机接到呼叫



的次数、机器出现的故障次数等。泊松分布的概率质量函数为:

$$P(X=k)=e^{-\lambda} \frac{\lambda^k}{k!}, k=0,1,2,\dots$$

若  $X$  服从参数为  $\lambda$  的泊松分布, 记为  $X \sim \pi(\lambda)$ , 或记为  $X \sim P(\lambda)$ 。

在了解了概率基础知识后, 再来看看连续型随机变量。对于随机变量  $X$ , 若存在非负可积函数  $p(x)(-\infty < x < +\infty)$ , 使得对任意实数  $a, b(a < b)$ , 都有  $P(a < X \leq b) = \int_a^b p(x)dx$ , 则  $X$  为连续型随机变量,  $P(a < X \leq b)$  为累积分布函数 (概率密度函数的积分, 可完整描述实随机变量  $X$  的概率分布), 函数  $p(x)$  为随机变量的概率密度函数,  $p(x)$  的图像为概率密度曲线, 如图 5-1 所示。

概率密度函数  $p(x)$  有如下性质:

$$\square p(x) \geq 0$$

$$\square \int_{-\infty}^{+\infty} p(x)dx = 1$$

定积分的几何意义为: 对于一个给定的正实值函数  $f(x)$ ,  $f(x)$  在一个实数区间  $[a, b]$  上的定积分  $\int_a^b f(x)dx$  是在  $Oxy$  坐标平面上, 由曲线  $(x, f(x))$ 、直线  $x=a$ ,  $x=b$  以及  $x$  轴围成的曲边梯形的面积值, 如图 5-2 中的阴影部分。

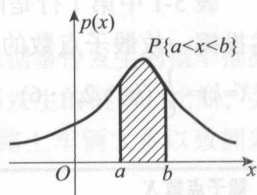


图 5-1 概率密度曲线

根据图 5-2 所示的定积分几何意义,  $\int_a^b p(x)dx$  为图 5-1 中的阴影部分, 其中  $p(x)$  为概率密度函数, 而  $P(a < X \leq b) = \int_a^b p(x)dx$ , 因此, 连续随机变量  $X$  在  $a$  与  $b$  之间分布的概率为概率密度函数在区间  $[a, b]$  上的定积分。

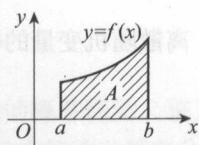


图 5-2 定积分的几何意义

概率质量函数和概率密度函数不同之处在于: 概率质量函数针对离散随机变量定义, 本身代表随机变量在确定取值点处的概率; 概率密度函数针对连续随机变量定义, 代表随机变量在确定取值点附近可能性的概率, 只有对连续随机变量的概率密度函数在某区间内进行积分 (积分后得到累积分布函数) 后才能完成概率的计算。

6) 连续型均匀分布。连续型随机变量在等长度的每个空间上取值的概率都相同, 则称  $X$  服从  $[a, b]$  上的均匀分布, 记作  $X \sim U[a, b]$ , 它的概率密度函数为:

$$f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & \text{其他} \end{cases}$$

累积分布函数为:

$$F(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & x \geq b \end{cases}$$

连续型均匀分布函数的期望值和中值等于区间  $[a, b]$  上的中间点:

$$E[X] = \frac{a+b}{2}$$

方差为:

$$\text{Var}[X] = \frac{(b-a)^2}{12}$$

7) 指数分布。指数分布可用来表示独立随机事件发生的时间间隔, 比如旅客进机场的时间间隔等。它的概率密度函数随着取值的变大而指数减小, 定义如下:

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0, & \text{其他} \end{cases}$$

其中,  $\lambda$  是指每单位时间发生该事件的次数,  $\lambda > 0$ 。

累积分布函数定义为:

$$F(x) = \begin{cases} 0, & x < 0 \\ 1 - e^{-\lambda x}, & x \geq 0 \end{cases}$$

8) 正态分布。若随机变量  $X$  服从一个位置参数为  $\mu$ 、尺度参数为  $\sigma$  的概率分布, 记为:  $X \sim N(\mu, \sigma^2)$ , 服从正态分布的随机变量的概率规律为: 与  $\mu$  越邻近的值的概率越大, 离  $\mu$  越远的值的概率越小;  $\sigma$  越小, 分布越集中在  $\mu$  附近,  $\sigma$  越大, 分布越分散。正态分布的概率密度函数曲线呈钟形, 又经常被称为钟形曲线, 密度函数为:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, -\infty < x < +\infty$$

累积分布函数为:

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt$$

正态分布密度具有如下性质:

□ 曲线关于直线  $x = \mu$  对称。

□ 当  $x = \mu$  时,  $f(x)$  取得最大值  $\frac{1}{\sqrt{2\pi}\sigma}$ ,

□  $f(x)$  有渐近线  $y = 0$ 。

□  $f(x)$  有两个拐点  $\left(\mu \pm \sigma, \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}}\right)$ 。

□  $\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dx = 1$ 。

### 3. 随机变量数字特征相关公式

分布函数完全刻画了随机变量取值的概率规律, 因此, 在实践中, 我们需要知道随机变量的某些分布指标, 比如变量的分布趋势、分布均匀程度等, 以便分析随机变量的数字特征。

1) 数学期望。数学期望反映了随机变量的平均取值。离散性随机变量的数学期望是试验中每次可能结果的概率乘以其结果的总和, 离散型随机变量  $X$  的数学期望为:

$$E[X] = \sum_{k=1}^{+\infty} x_k p_k$$

连续型随机变量  $X$  的数学期望为:

$$E[X] = \int_{-\infty}^{+\infty} x f(x) dx$$

2) 方差。方差刻画了随机变量对它的均值的偏离程度, 如果  $E[X]$  是随机变量  $X$  的期望值 (平均数  $\mu=E[X]$ ), 则随机变量  $X$  或者分布  $F$  的方差为:

$$\text{Var}(X)=E[(X-\mu)^2]$$

3) 协方差。协方差表示的是两个变量的总体的误差, 如果两个变量的变化趋势一致, 也就是说, 如果其中一个大于自身的期望值, 另外一个也大于自身的期望值, 那么两个变量之间的协方差就是正值。如果两个变量的变化趋势相反, 即其中一个大于自身的期望值, 另外一个却小于自身的期望值, 那么两个变量之间的协方差就是负值。如果  $X$  与  $Y$  是统计独立的, 那么二者之间的协方差就是 0。

期望值分别为  $E(X)=\mu$  与  $E(Y)=v$  的两个实数随机变量  $X$  与  $Y$  之间的协方差, 定义为:

$$\text{cov}(X, Y)=E[(X-\mu)(Y-v)]$$

其中  $E$  是期望值, 也可表示为:

$$\text{cov}(X, Y)=E(XY)-\mu v$$

## 5.3 回归分析

在进行回归分析时, 通常利用数理统计中的回归方法, 确定两种或两种以上变量间相互依赖的定量关系。

### 5.3.1 单变量线性回归

在第 3 章中曾谈到数据的线性回归, 可能会出现一种情况, 所有的数据点都准确地落在了回归线上。但在现实中, 很难有如此精确的模型, 比如有两个变量  $x$  和  $y$ , 其中,  $x=[5,7,9,11,16,20]$ ,  $y=[1,2,3,4,7,9]$ , 现在要在  $x$  与  $y$  之间建立回归模型, 该如何实现?

先用 R 语言构造数据点。

```
> y<-c(5,7,9,11,16,20)
> x<-c(1,2,3,4,7,9)
```

在建立回归线之前, 先通过 R 语言的 plot 函数绘制这些点的位置, 观察分布规律, 如图 5-3 所示。

```
> plot(x,y)
```

图 5-3 称为散点图, 能清晰地坐在坐标系中查看数据点位置, 横轴为  $x$ , 纵轴为  $y$ 。

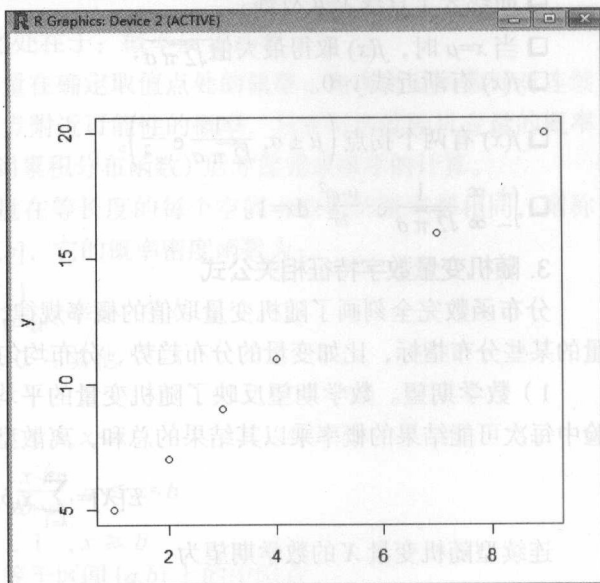


图 5-3 散点图

从散点图上可以看出  $x$  和  $y$  之间存在线性关系, 可以表示为  $y=kx+b$  这种一次函数的模型。不过想要画一条直线完全穿过这些点是不可能的, 但能保证这些点到这条直线的距离最小, 这个距离就是残差。残差是观测值与预测值之间的差。下面来具体看一下。

首先, 通过 `lsfit` 函数计算回归直线方程的斜率和截距以及残差。代码如下:

```
> lsfit(x,y)
$coefficients
Intercept      X
  3.338028  1.845070
$residuals
[1] -0.18309859 -0.02816901  0.12676056  0.28169014 -0.25352113  0.05633803
```

上面代码中出现的 `residuals` 表示残差, 残差分别反映了这些点与直线的差异, 残差越小越好。残差直接反映了数据点到回归直线的距离,  $-0.18309859$ 、 $-0.02816901$ 、 $0.12676056$ 、 $0.28169014$ 、 $-0.25352113$ 、 $0.05633803$  分别是当  $x$  值为  $[1,2,3,4,7,9]$  时, 用  $y=1.845070x+3.338028$  回归方程求得的  $y$  值与实际  $y$  值  $[5,7,9,11,16,20]$  的差额。残差  $e_i$  的计算公式为:

$$e_i = y_i - \hat{y}_i (i=1,2,\dots,n)$$

其中,  $y_i$  表示实际值,  $\hat{y}_i$  为按回归方程预测的值。

例如, 将  $x=2$  代入该例中的回归方程, 即为  $y=1.845070 \times 2+3.338028$ , 计算可得到  $y$  的预测值, 然后得到残差, 如下所示:

实际  $y$  值 - 预测  $y$  值  $\approx 1.845070 \times 2+3.338028-7 \approx -0.02816901$

然后, 绘制散点图及回归线。命令如下:

```
> abline(lsfit(x,y))
```

刚才将 2 代入回归方程后, 计算  $1.845070 \times 2+3.338028-7$ , 得到残差仅为  $-0.02816901$ , 在图 5-4 中找到回归线上  $x=2$  时  $y$  的值, 能看出, 圆圈代表的  $y$  值几乎贴近回归线。再看  $x=1$ 、3、4、7、9 时, 回归线对应的  $y$  值都比较贴近回归线, 这些数据点紧靠在回归线周围, 显然回归效果不错。

残差不应该有某种趋势, 若残差中出现一种明显的趋势, 则意味着模型不适合。如图 5-4 所示的残差分布较均匀, 没有明显的趋势显示大量数据点偏离了回归线。

事实上, 也可以使用 `lm` 函数进行更详细的回归分析。

代码如下:

```
> x<-c(1,2,3,4,7,9)
> y<-c(5,7,9,11,16,20)
> lm(y~x)->xy
> summary(xy)
```

Call:

```
lm(formula = y ~ x)
```

Residuals:

```
1      2      3      4      5      6
```

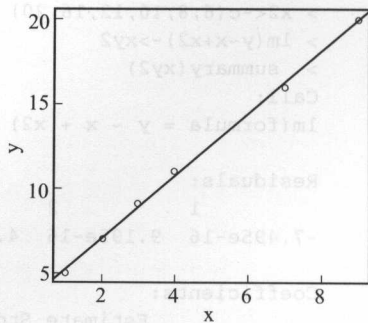


图 5-4 散点图及回归线

```
-0.18310 -0.02817 0.12676 0.28169 -0.25352 0.05634
```

```
Coefficients:
```

```
      Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.33803    0.16665   20.03 3.67e-05 ***
x            1.84507    0.03227   57.17 5.60e-07 ***
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.222 on 4 degrees of freedom
```

```
Multiple R-squared:  0.9988,    Adjusted R-squared:  0.9985
```

```
F-statistic: 3269 on 1 and 4 DF,  p-value: 5.604e-07
```

```
>plot(x,y)
```

```
>abline( lm(y~x) )
```

在上述代码中，Coefficients 栏的内容如下。

❑ Estimate: 斜率与截距的估计值。

❑ Std. Error: 斜率与截距的估计标准差。

❑ t value: 斜率与截距的假设检验的 t 值。

❑ Pr(>|t|): 与显著性水平比较，决定是否接受该假设检验。

在 Coefficients 每行的最后一列有星号，这个星号的含义表示线性关系是否显著：\* 的数量是 0 ~ 3，\* 的数量越多则线性关系越显著。本例中，Coefficients 栏的 Pr(>|t|) 字段（3.67e-05 以及 5.60e-07）后标注有 \*\*\*，说明 x 与 y 之间的线性关系很强。

### 5.3.2 多元线性回归

多元性回归可建立多个自变量和应变量之间的关系，其回归模型方程一般为：

$$y=b_0+b_1x_1+b_2x_2+\cdots+b_kx_k+e$$

其中，y 为因变量， $x_1, x_2, \cdots, x_k$  为自变量， $b_0$  为常数项， $b_1, b_2, \cdots, b_k$  为回归系数。

在进行多元线性回归分析时，可使用 lm 函数，在上节例子的基础上增加一个自变量  $x_2=[3,4,5,6,8,10]$ ，来看看会有什么效果。代码如下：

```
> y<-c(5,7,9,11,16,20)
> x<-c(1,2,3,4,7,9)
> x2<-c(6,8,10,12,16,20)
> lm(y~x+x2)->xy2
> summary(xy2)
Call:
lm(formula = y ~ x + x2)
```

```
Residuals:
```

```
      1      2      3      4      5      6
-7.495e-16  9.195e-16  4.172e-17 -2.117e-16  1.839e-16 -1.839e-16
```

```
Coefficients:
```

```
      Estimate Std. Error  t value Pr(>|t|)
(Intercept)  1.000e+00  3.787e-15  2.640e+14  <2e-16 ***
```



```

x          1.000e+00  1.359e-15  7.357e+14  <2e-16 ***
x2         5.000e-01  8.019e-16  6.236e+14  <2e-16 ***
---

```

```

Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

Residual standard error: 7.121e-16 on 3 degrees of freedom

```

```

Multiple R-squared: 1, Adjusted R-squared: 1

```

```

F-statistic: 1.591e+32 on 2 and 3 DF, p-value: < 2.2e-16

```

通过上述回归分析, 得出该回归方程为  $y=x+0.5 \times x_2+1$ , Std. Error、t value、Pr(>|t|) 以及线性相关程度等指标的含义同单变量线性回归类似。

### 5.3.3 非线性回归

非线性回归模型较多, 其中应用得较多的有以下模型:

#### 1) 多项式模型:

$$y=\beta_0+\beta_1x+\beta_2x^2+\cdots+\beta_kx^k+\varepsilon$$

#### 2) 指数模型:

$$y=ae^{bx}$$

#### 3) 幂函数模型:

$$y=ax_1^{b_1}x_2^{b_2}\varepsilon$$

#### 4) 成长曲线模:

$$y=1/(\beta_0+\beta_1e^{-x}+\varepsilon)$$

实际应用中, 除上述模型外, 还有很多非线性回归模型, 但无论是哪种非线性回归模型, 最后都可以通过变量变换转化为线性模型, 从而用最小二乘法进行回归分析。下面以  $y=\beta_0+\beta_1e^{bx}+\varepsilon$  模型为例讲解非线性回归的方法。

#### 1) 准备回归分析用数据 (假设已预先知道回归方程, 通过回归方程精确计算 y 值),

```

>x<-c(1,2,3,4,7,8,9)

```

```

>y <- 100 + 10 * exp(x / 2) + rnorm(x)

```

#### 2) 使用 R 语言的 nls 函数, 应用最小二乘法原理, 实现非线性回归分析。

```

>nlsmod <- nls(y ~ Const + A * exp(B * x))

```

#### 3) 使用 summary 函数分析拟合结果。

```

> summary(nlsmod)

```

```

Formula: y ~ Const + A * exp(B * x)

```

```

Parameters:

```

```

      Estimate Std. Error t value Pr(>|t|)

```

```

Const 1.001e+02  8.377e-01  119.4 2.95e-08 ***

```

```

A      1.006e+01  1.759e-01   57.2 5.59e-07 ***

```

```

B      4.995e-01  1.925e-03  259.5 1.32e-09 ***

```

```

--- Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 1.1 on 4 degrees of freedom
Number of iterations to convergence: 8
Achieved convergence tolerance: 4.887e-07
>

```

其中，在 Parameters 栏，对方程涉及的以下 3 个参数进行了预测：

$$\beta_0 = \text{Const} = 1.001\text{e}+02$$

$$\beta_1 = A = 1.006\text{e}+01$$

$$b = B = 4.995\text{e}-01$$

将以上参数与程序中  $y$  值的生成规则进行对比，可以看见，拟合效果还是不错的。

前面为解释非线性回归过程，将  $x$  代入参数确定的非线性回归方程计算  $y$  值，然后，依据  $x$  和  $y$  推出非线性回归方程的参数。但实践应用中，往往需要依据一组  $x$  值和  $y$  值，推导非线性方程的参数，因此，尝试通过 `rnorm` 函数产生较小的随机数，加在精确计算的  $y$  值上，这样计算后形成的非线性回归模型拥有一定的残差，较接近真实环境。

```
>y <- 100 + 10 * exp(x / 2) + rnorm(x)
```

4) 绘制拟合效果图。代码如下：

```

>plot(x,y, main = "nls(o)")
>curve(100 + 10 * exp(x / 2), col = 4, add = TRUE)
>lines(x, predict(nlmod), col = 2,type='b')

```

如图 5-5 所示为拟合效果图，显示出拟合效果不错。其中，偏上的线为预测的回归线，偏下的线为实际方程。回归在 [4,7] 的区间内拟合效果较差，这是样本数据太少的原因，因为样本数据仅有 9 个。

参与拟合的样本数据量决定了拟合效果的好坏。可以加大自变量的数量，重新应用 `nls` 函数进行拟合，来看看效果如何。代码如下：

```

>x<-seq(1,10,0.1)
>y <- 100 + 10 * exp(x / 2) + rnorm(x)
>nlmod <- nls(y ~ Const + A * exp(B * x))
>plot(x,y, main = "nls(o)")
>curve(100 + 10 * exp(x / 2), col = 4, add = TRUE)
>lines(x, predict(nlmod), col = 2)

```

如图 5-6 所示是相应的效果图，从中可以看出，回归线与实际方程线很接近。

不过，图 5-6 所示的回归效果仍存在一个问题，就是样本过于集中在回归线上了，为使回归分析更接近真实应用环境，需要继续加大随机数的范围，增加非线性回归的残差，使样本点散布在回归线周围。

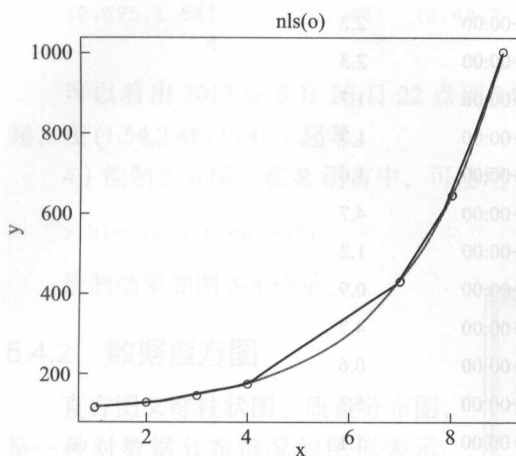


图 5-5 拟合效果图

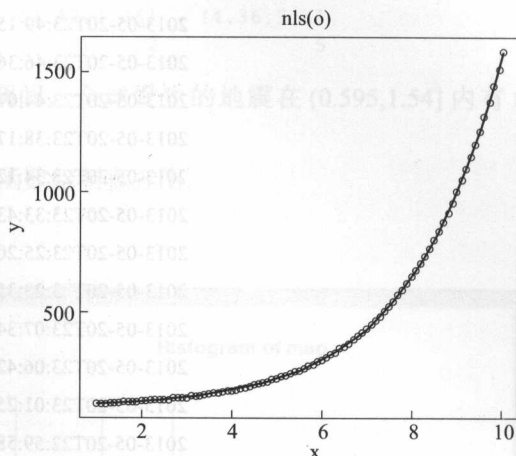


图 5-6 非线性回归效果图

```
>x<-seq(1,10,0.1)
>y <- 100 + 10 * exp(x / 2) + rnorm
(x)*100
>nlsmod <- nls(y ~ Const + A * exp(B *
x))
>plot(x,y, main = "nls(o)")
>curve(100 + 10 * exp(x / 2), col = 4,
add = TRUE)
>lines(x, predict(nlsmod), col = 2)
```

从图 5-7 可以看出，样本点虽然没有集中在回归线上，而是散落在回归线周围的区域，产生的残差较大，但样本点的整体走向与回归线一致，此外，回归线与实际方程这两条线几乎重叠，说明回归分析较准确地预测出回归方程的各个参数。因此，从整体上观察，拟合效果不错，回归模型适当。

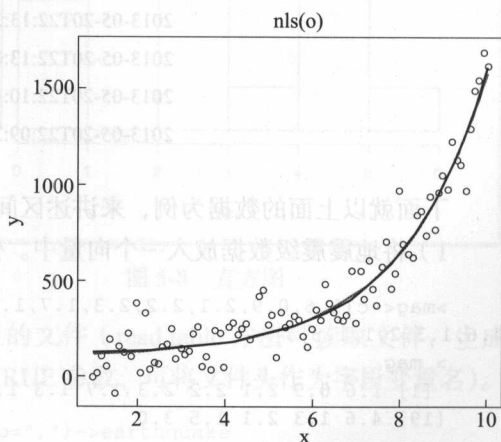


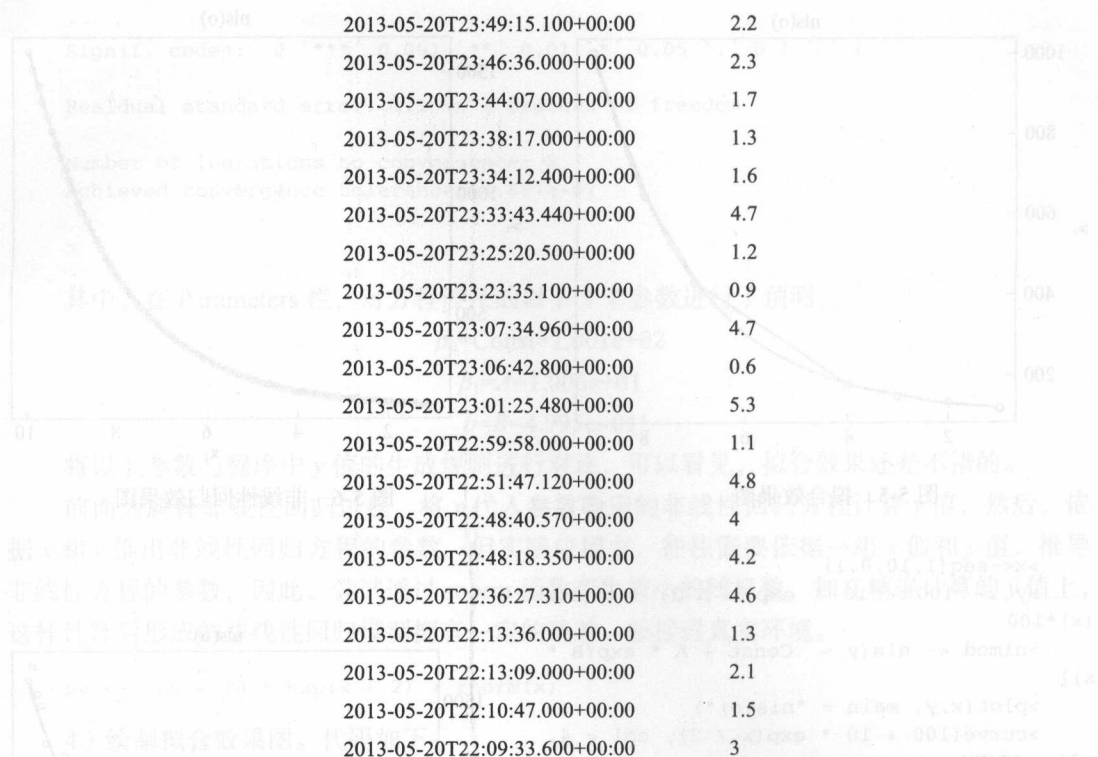
图 5-7 接近真实环境的非线性回归

## 5.4 数据分析基础

### 5.4.1 区间频率分布

下面是美国地震台网公布的全球 2013 年 5 月 20 日 22 点到 24 点发生的所有地震的震级。

时间	震级
2013-05-20T23:57:12.000+00:00	1.6
2013-05-20T23:57:12.000+00:00	0.9
2013-05-20T23:52:59.000+00:00	2.1



下面就上面的数据为例，来讲述区间频率分布。现在的任务是完成地震震级分析。

1) 将地震震级数据放入一个向量中。代码如下：

```
>mag<-c(1.6,0.9,2.1,2.2,2.3,1.7,1.3,1.6,4.7,1.2,0.9,4.7,0.6,5.3,1.1,4.8,4.4,2.2,
4.6,1.3,2.1,1.5,3)
> mag
[1] 1.6 0.9 2.1 2.2 2.3 1.7 1.3 1.6 4.7 1.2 0.9 4.7 0.6 5.3 1.1 4.8 4.0 4.2
[19] 4.6 1.3 2.1 1.5 3.0
```

2) 使用 cut 函数将震级分成 5 个区间，并建立因子。代码如下：

```
>factor(cut(mag,5))
[1] (1.54,2.48] (0.595,1.54] (1.54,2.48] (1.54,2.48] (1.54,2.48]
[6] (1.54,2.48] (0.595,1.54] (1.54,2.48] (4.36,5.3] (0.595,1.54]
[11] (0.595,1.54] (4.36,5.3] (0.595,1.54] (4.36,5.3] (0.595,1.54]
[16] (4.36,5.3] (3.42,4.36] (3.42,4.36] (4.36,5.3] (0.595,1.54]
[21] (1.54,2.48] (0.595,1.54] (2.48,3.42]
Levels: (0.595,1.54] (1.54,2.48] (2.48,3.42] (3.42,4.36] (4.36,5.3]
```

3) 统计因子频率。代码如下：

```
>factor(cut(mag,5))->magfactor
> table(magfactor)
magfactor
```

```
(0.595,1.54] (1.54,2.48] (2.48,3.42] (3.42,4.36] (4.36,5.3]
      8           7           1           2           5
```

可以看出 2013 年 5 月 20 日 22 点到 24 点期间, 全球发生的地震在 (0.595,1.54] 内有 8 起, 在 (1.54,2.48] 内有 7 起等。

4) 绘制直方图。在 R 语言中, 可使用 hist 函数绘制直方图。

```
> hist(mag,breaks=5)
```

绘制结果如图 5-8 所示。

## 5.4.2 数据直方图

直方图又称柱状图、质量分布图, 是一种对数据分布情况的图形表示, 由一系列高度不等的纵向条纹或线段表示数据分布的情况, 它根据从生产过程中收集来的质量数据分布情况, 组成以组距为底边、以频数为高度的一系列连接起来的直方型矩形图。

地震数据是一种次数直方图, 是由若干宽度相等、高度不一的直方条紧密排列在同一基线上构成的图形, 其中基线上每个区间代表了一段震级, 高度代表了这段震级发生地震的次数(频率)。

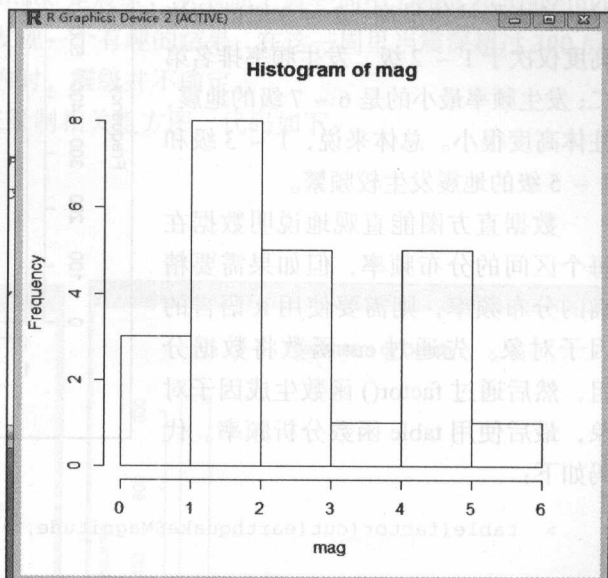


图 5-8 直方图

1) 用 R 语言的 read.table 函数读取地震震级的文件 (read.table 方法可读取文件, 生成 list 组件类型的数据集, 并且, 通过指定 header=TRUE 参数, 可将文件头作为字段变量名)。

```
> read.table("eqweek.csv",header=TRUE,sep=",")->earthquake
```

2) 显示读取的地震数据 (2013.5.14 ~ 2013.5.20), 验证读取内容是否完整。

```
> earthquake      DateTime.Latitude.Longitude.Depth.Magnitude.MagType.
NbStations.Gap.Distance.RMS.Source.EventID.Version
1      2013-05-20T23:57:12.000+00:00,63.45,-148.291,5.5,1.6,M1,,,,
0.8,ak,ak10720946,1.3691E+12
2      2013-05-20T23:52:59.000+00:00,61.337,-152.069,81.4,2.1,M1,,,,
1.15,ak,ak10720941,1.36909E+12
3      2013-05-20T23:49:15.100+00:00,19.99,-155.426,38.2,2.2,Md,,133,0.1,0.
11,hv,hv60501711,1.3691E+12
4      2013-05-20T23:46:36.000+00:00,60.498,-142.974,4.2,2.3,M1,,,,
0.43,ak,ak10720934,1.36909E+12
.....
```

3) 绘制数据直方图, 观察从 2013 年 5 月 14 日至 2013 年 5 月 20 日这周内全球地震震



级的分布情况。代码如下：

```
> hist(earthquake$Magnitude,5)
```

生成的直方图如图 5-9 所示。其中横轴是震级，纵轴是频率分析。通过观察可得出结论：震级在 1 ~ 2 级的地震发生频率最高，因为在  $x$  为 1 ~ 2 时，长方形柱体最高；2 ~ 3 级的柱体高度仅次于 1 ~ 2 级，发生频率排名第二；发生频率最小的是 6 ~ 7 级的地震，柱体高度很小。总体来说，1 ~ 3 级和 4 ~ 5 级的地震发生较频繁。

数据直方图能直观地说明数据在每个区间的分布频率，但如果需要精确的分布频率，则需要使用 R 语言的因子对象。先通过 cut 函数将数据分组，然后通过 factor() 函数生成因子对象，最后使用 table 函数分析频率。代码如下：

```
> table(factor(cut(earthquake$Magnitude,5)))
```

(0.995,2.1]	(2.1,3.2]	(3.2,4.3]	(4.3,5.4]	(5.4,6.51]
693	200	46	126	10

观察上述结果的最后两行，每个区间的震级频率一目了然，(0.995,2.1] 区间的震级频率是 693 次，(2.1,3.2] 区间的震级频率是 200 次等。

### 5.4.3 数据散点图

数据散点图是指数据点在直角坐标系平面上的分布图，通过直接观察图形辨认某现象的测量值与可能原因因素之间的关系，具有快捷、易于交流和易于理解的特点。

数据散点图表示因变量随自变量而变化的大致趋势，将序列显示为一组点，值由点在图表中的位置表示，类别由图表中的不同标记表示。通常用垂直轴表示现象值  $Y$ ，用水平轴表示可能有关系的原因因素  $X$ ，通过对其观察分析，来判断两个变数之间的相关关系。此外，依据散点图可选择函数对数据点进行拟合，建立回归模型。

下面继续以全球一周地震数据为例，来讲解数据散点图。

1) 将变量放到搜索路径上。代码如下：

```
> attach(earthquake)
```

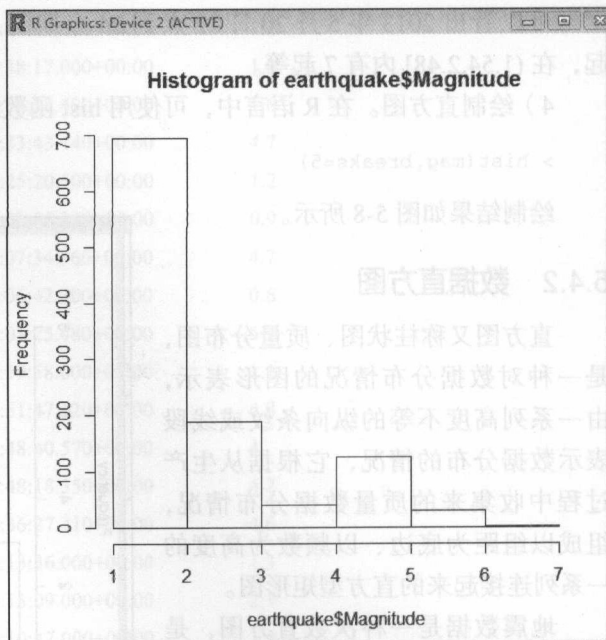


图 5-9 震级分布直方图

## 2) 分析地震震深。代码如下:

```
> summary (Depth)
Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
0.10 5.80 12.15 30.82 38.00 630.70 39
```

在上述结果中, Min 表示地震震深的最小值, Max 表示最大值, Median 为中位数, Mean 为平均值。我们试着从下面的散点图中分析一下地震震深与震级的关系, 如图 5-10 所示。

在图 5-10 中, Depth 是震深, Magnitude 是震级, 从表面上看一周中 Depth 和 Magnitude 之间没有关系, 但仔细观察这个图, 可发现一个有趣的结果: 在这一周里当震深超过 300 后, 震级都接近 5 或在 5 以上, 而在 300 以内时, 震级并不确定。

## 3) 对震深的直方图进行分析。先来绘制相关直方图, 代码如下:

```
> hist (Depth)
```

绘制的图形如图 5-11 所示。

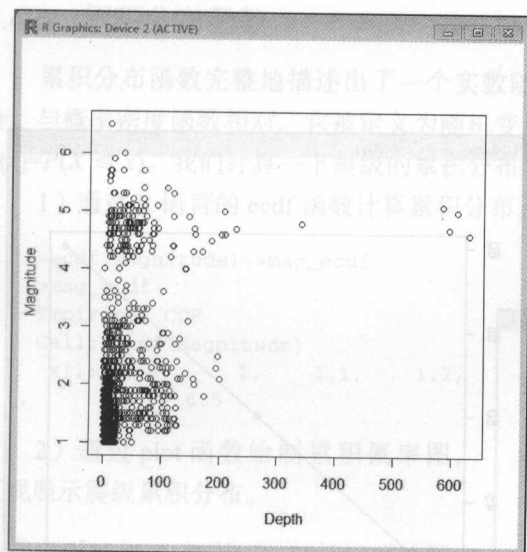


图 5-10 震深与震级关系的散点图

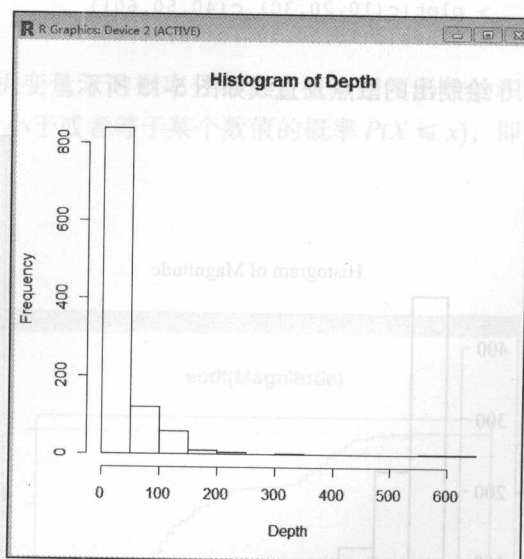


图 5-11 震深的直方图

观察图 5-11 可得出结论: 这个星期内发生的绝大部分地震的震深在 100 以内, 这个区间的代表频率的柱体高度最高, 而发生最少的 350 ~ 550 段和 250 ~ 300 段, 这两个区间的柱体几乎贴近 X 轴, 高度很小。

4) 分析带数据点的震级直方图。为提高直方图的表示能力, 有时需要在直方图中显示实际的数据点, 通过观察这些数据点的数量, 可分析数据点在每个区间的分布密集程度, 可通过 R 语言的 rug 函数绘制数据点。代码如下:

```
> hist (Magnitude)
```

```
> rug(Magnitude)
```

绘制出的图形如图 5-12 所示。

仔细观察图 5-12 能看出：在 3 ~ 4 震级区域内，数据点分布并不均匀，在靠近 3 的区域内，数据更为密集，数据点集中分布在偏向于 3 的区域；此外，在 5 ~ 6 级震级区域也出现类似现象，数据点偏向于 5，这说明 3 ~ 4 级或 5 ~ 6 级震级范围内，大部分地震的震级不是非常大，偏向于 3 或 5。

这些只是根据一个星期的数据进行分析得到的结果，不一定就代表真正的答案。答案要通过分析大量数据以及数据各个指标关系，同时结合地球物理知识的研究才能得出。

由于图形直观性很强，浅显且易于理解，因此在数据分析中，经常需要绘制图表和直线。前面已经讲解过绘图的方法，在此补充一下画线方法，在 R 语言中使用 lines 函数完成绘制直线。比如要绘制一个 (10,40)、(20,50)、(30,60) 的散点图，并将点连成线，可如下编写代码：

```
> plot(c(10,20,30),c(40,50,60))
> lines(c(10,20,30),c(40,50,60))
```

绘制出的散点及直线如图 5-13 所示。

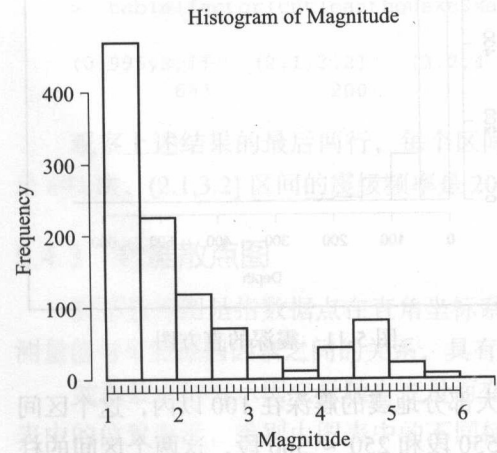


图 5-12 带数据点的震级直方图

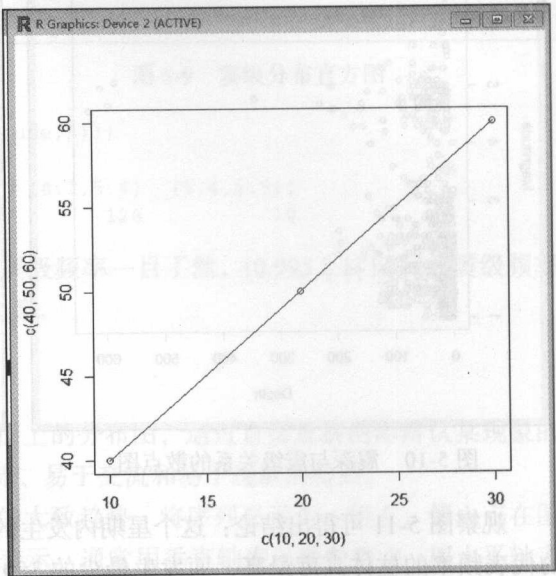


图 5-13 散点及直线图

### 5.4.4 五分位数

分位数是描述数据位置的一种方法，它将一个随机变量的分布范围分为几个等份的数值点。分位数法被用来识别某临界值，一般情况下可使用分位数（或分位点）描述小于等于这个

临界值的观测值数量占整个数据中的某个具体比率（小于等于该值的数据为一给定的比率）。

常用的分位数有中位数、四分位数、五分位数、十分位数、百分位数等，其中，中位数将数据分布范围分成了相等的两个部分。此外，我们还能将数据分布范围分成更小尺寸的分割线，四分位数将一个分布分成 4 等份，而五分位数将其分成 5 等份，十分位数将其分成 10 等份，百分位数将其分成 100 等份等。

五分位数法是数据分析的常用方法，在 R 语言中，可使用 `fivenum` 函数计算五分位数（`fivenum` 函数会返回以下数据：minimum、lower-hinge、median、upper-hinge、maximum）。下面用 `fivenum` 函数对地震数据进行分析。代码如下：

```
> fivenum(Magnitude)
[1] 1.0 1.3 1.7 2.5 6.5
```

分析结果表明，震级最小为 1.0，最大为 6.5，中位数为 1.7，通过 1.7 将一组数据分为上下两组，然后再计算上下两组的中位数 1.3 与 2.5。

#### 5.4.5 累积分布函数

累积分布函数完整地描述出了一个实数随机变量  $X$  的概率分布，是概率密度函数的积分，与概率密度函数相对。它被定义为随机变量小于或者等于某个数值的概率  $P(X \leq x)$ ，即  $F(x) = P(X \leq x)$ 。我们计算一下震级的累积分布。

1) 通过 R 语言的 `ecdf` 函数计算累积分布。

```
> ecdf(Magnitude) -> mag_ecdf
> mag_ecdf
Empirical CDF
Call: ecdf(Magnitude)
x[1:50] = 1, 1.1, 1.2, ..., 6, 6.5
```

2) 通过 `plot` 函数绘制累积概率图，直观展示震级累积分布。

```
> plot(mag_ecdf, do.points=FALSE,
verticals=TRUE)
```

绘制出的图形如图 5-14 所示。

图 5-14 的  $x$  轴是震级， $y$  轴是累积分布概率，当震级升高时，累积分布概率也随之提高，这个趋势很正常，因为累积分布概率是指小于或等于某个震级的概率。比如：从图 5-14 观察，发生地震的震级在 3 以内的概率接近 80%，震级在 2 以内的概率接近 60%。

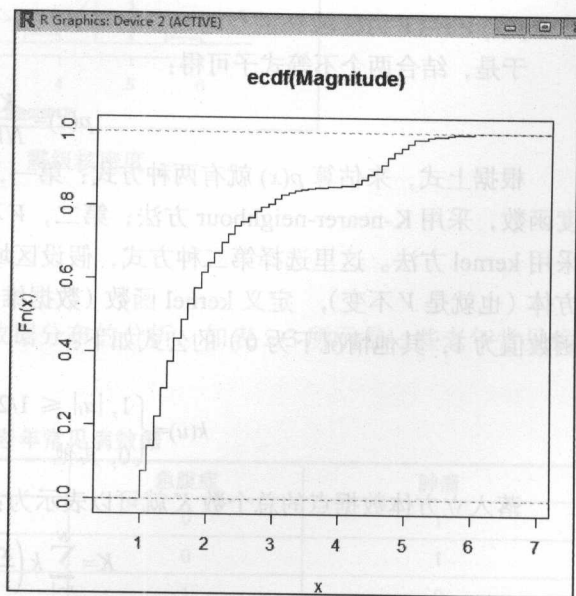


图 5-14 震级累积分布概率



## 5.4.6 核密度估计

### 1. 核密度原理

利用前面所说的直方图估计数据分布密度是有局限性的, 因为数据分布的密度函数是不平滑的, 它受子区间宽度影响较大, 当数据维数超过二维时就有局限。

什么是密度函数呢? 连续型随机变量的概率密度函数 (不至于混淆时可简称为密度函数) 是一个描述这个随机变量的输出值, 在某个确定的取值点附近的可能性的函数, 而随机变量落在某个区域之内的概率为概率密度函数在这个区域上的积分, 当概率密度函数存在的时候, 累积分布函数是概率密度函数的积分。

如何由给定样本点集合求解随机变量的分布密度函数? 解决这一问题的方法包括参数估计和非参数估计。参数估计中常用的是参数回归分析, 它假定数据分布符合某种特定的性态, 如线性、可化线性或指数性态等, 然后确定回归模型的未知参数。但参数模型的这种基本假定与实际的物理模型之间常常存在较大的差距, 于是 Rosenblatt 和 Parzen 提出了核密度估计方法, 它可估计未知的密度函数, 是非参数估计方法之一。

假设样本数据值在  $D$  维空间服从一个未知的概率密度函数, 那么在区域  $R$  内的概率为:

$$P = \int_R p(x) dx$$

上式中,  $P$  是每个样本数据点落入区域  $R$  的概率, 假设在  $N$  个样本数据点中, 有  $K$  个落入了区域  $R$ , 那么就应该服从二项分布。公式如下:

$$\text{Bin}(K|N, P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{N-K}$$

在  $N$  的样本数据很大时,  $K$  约等于  $N \times P$ 。而另一方面, 假设区域  $R$  足够小, 那么  $P$  约等于  $p(x) \times V$  ( $V$  为区域  $R$  的空间)。

于是, 结合两个不等式子可得:

$$p(x) = \frac{K}{NV} \quad (5-1)$$

根据上式, 来估算  $p(x)$  就有两种方式: 第一,  $K$  不变, 通过决定区域  $V$  的大小来估算密度函数, 采用 K-nearer-neighbour 方法; 第二,  $V$  不变, 通过决定  $K$  的大小来估算密度函数, 采用 kernel 方法。这里选择第二种方式, 假设区域  $R$  是一个以  $x$  为中心、边长为  $h$  的极小立方体 (也就是  $V$  不变), 定义 kernel 函数 (数据维数为  $D$  维, 当样本数据点落入小立方体时, 函数值为 1, 其他情况下为 0) 的公式如下:

$$k(u) = \begin{cases} 1, & |u_i| \leq 1/2, i=1, \dots, D \\ 0, & \text{其他} \end{cases}$$

落入立方体数据点的总个数  $K$  就可以表示为:

$$K = \sum_{n=1}^N k\left(\frac{x-x_n}{h}\right) \quad (5-2)$$

根据式 (5-1), 把式 (5-2) 代入式 (5-1) 中, 可得:



$$p(x) = \frac{1}{N} \sum_{n=1}^N \frac{1}{h^D} k\left(\frac{x-x_n}{h}\right)$$

上式中,  $V=h^D$ 。

## 2. 核密度计算

R 语言可使用 density 函数进行核密度估计, density 函数默认情况下在 512 个点上估计密度值。下面就用它来计算一下地震数据中的震级核密度。首先, 指定 hist 函数的 prob 参数为 TRUE, 绘制核密度直方图; 然后通过 lines 函数, 绘制核密度曲线。代码如下:

```
> hist(Magnitude, prob=TRUE)
> lines(density(Magnitude))
```

绘制出的核密度如图 5-15 所示, 图中曲线就是核密度曲线。

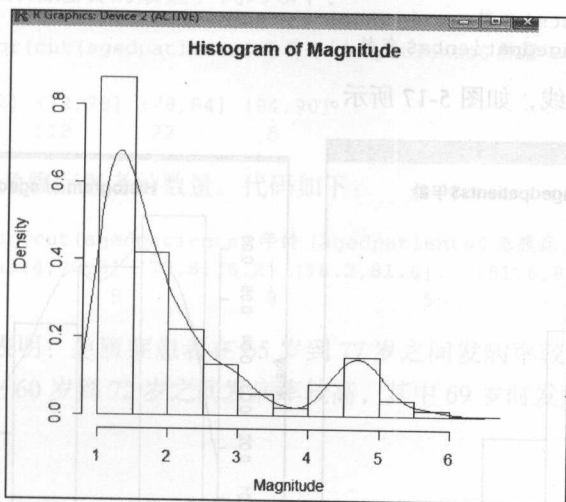


图 5-15 震级核密度

## 5.5 数据分布分析

这里以老年常见病数据为例, 来讲解数据分布的分析。如表 5-3 所示是一些老年常见病数据。

表 5-3 老年常见病数据

年龄	疾病名称	急腹症	肿瘤
71	肺恶性肿瘤	0	1
75	直肠恶性肿瘤	0	1
75	肠梗阻	1	0
71	急性胆管炎	1	0

(续)

年龄	疾病名称	急腹症	肿瘤
77	胆管恶性肿瘤	0	1
...	...	...	...

1) 在 R 中加载数据, 然后查看老年病的老人年龄分布情况及概率密度分布, 绘制年龄直方图。代码如下:

```
> read.table("aged_patients.csv",header=TRUE,sep=",")->agedpatients
> hist(agedpatients$ 年龄)
```

绘制出的年龄分布直方图如图 5-16 所示。

2) 对年龄进行核密度估计, 并绘制核密度曲线。

```
> hist(agedpatients$ 年龄 ,prob=TRUE)
> lines(density(agedpatients$ 年龄 ))
```

绘制出的核密度曲线, 如图 5-17 所示。

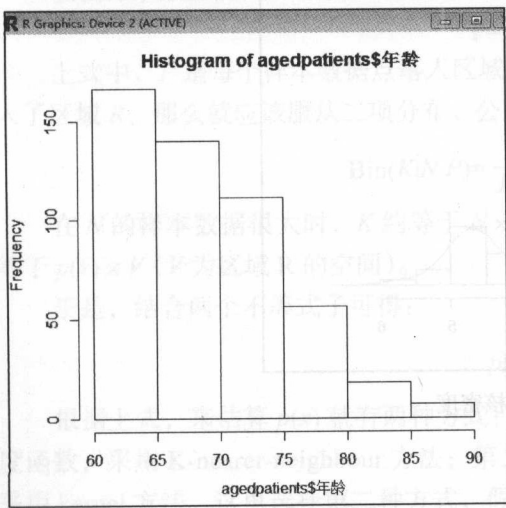


图 5-16 年龄分布直方图

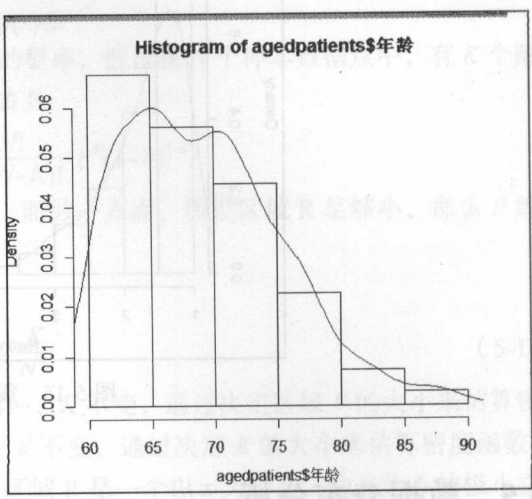


图 5-17 年龄的核密度曲线

3) 通过 min 和 max 两个函数分析患者的最小年龄和最大年龄, 通过 mean 函数分析年龄的平均值, 通过 var 函数计算年龄的方差。

```
> min(agedpatients$ 年龄 )
[1] 60
> max(agedpatients$ 年龄 )
[1] 90
> mean(agedpatients$ 年龄 )
[1] 69.09839
> var(agedpatients$ 年龄 )
[1] 38.60801
```

由上述分析可得出一个结论：63 ~ 72 岁这个阶段的老人必须要注意身体，坚持运动，保持健康，这个年龄段是急腹症和肿瘤两种老年病的高发期，发生概率较大。从 60 到 90 岁的老人都有可能患上这两种老年病，患者的平均年龄是 69 岁。标准差比较大，看来这两种老年病在老人的各个年龄段中发生得比较普遍。

4) 按年龄分类汇总肿瘤和急腹症的数量。代码如下：

```
> attach(agedpatients)
> tapply(肿瘤, 年龄, sum)
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 87 88 90
19 26 26 28 21 35 37 22 21 23 29 26 31 16 19 12 13 19 8 5 4 3 7 2 1 1 0 3 1
> tapply(急腹症, 年龄, sum)
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 87 88 90
2 0 0 2 0 0 1 0 1 2 4 1 1 0 3 1 2 2 0 2 0 1 0 1 0 1 2 0 0
```

5) 分年龄统计肿瘤患者的数量。代码如下：

```
> table(factor(cut(agedpatients$年龄 [agedpatients$肿瘤 ==1],5)))
(60,66] (66,72] (72,78] (78,84] (84,90]
155      158      118       22       5
```

6) 分年龄统计急腹症患者的数量。代码如下：

```
> table(factor(cut(agedpatients$年龄 [agedpatients$急腹症 ==1],5)))
(60,65.4] (65.4,70.8] (70.8,76.2] (76.2,81.6] (81.6,87]
4          8          8          5          4
```

上述分析结果表明：急腹症患者在 65 岁到 77 岁之间发病率较高，其中 70 岁时发病率最高；而肿瘤患者在 60 岁到 72 岁之间发病率较高，其中 69 岁时发病率最高。

## 5.6 小结

本章对统计学基础进行阐述，介绍了数据分析的概念、发展以及需要的基础知识，同时从线性回归和非线性回归两个方面讲述了数据分析基本方法——回归分析，并以实例说明了用 R 语言进行数据分布情况分析的方法。

统计分析在机器学习和数据分析中有着举足轻重的地位。李开复在攻读博士期间主攻语音识别。他的导师坚持的方向是发展和完善专家系统。而他最终发现，专家系统是有严重局限性的，无法延伸到做不特定语者的语音识别。他认为有数据支持的统计模式是唯一的希望，于是改用统计的方法来进行语音识别。3 年中，他用统计的方法把语音识别的准确率从 40% 逐步提高到 80%、90%，最后达到了 96%，《商业周刊》把他的发明选为 1988 年最重要的科学发明。他用统计学方法做出的语音识别博士论文至今还被用作语音识别产品的理论基础。

Google 在 2006 年面对广大用户推出了关键词统计分析系统：Google Trends，链接为：<http://www.google.com/trends>。这是一个非常有意思和价值的产品，有兴趣的读者不妨一试。下面是利用这个系统对“machine learning”这个搜索关键词进行的统计分析，从图 5-18 和

图 5-19 中可以明显看出，机器学习越来越受人关注，尤其是从 2011 年年底开始，人们对机器学习的热衷度在持续上升，因为这段时期的曲线呈稳步上扬趋势。

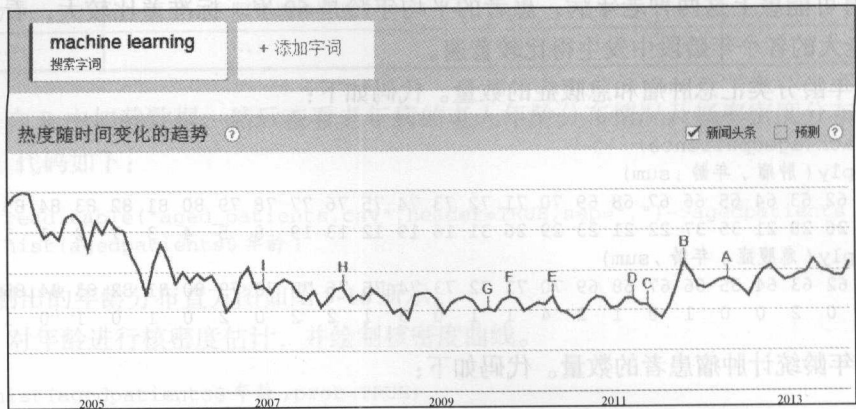


图 5-18 “机器学习”在 Google Trends 中的搜索结果

热门		上升	
the learning machine	100	google machine learning	飙升
pattern recognition	50	machine learning python	飙升
machine learning pdf	50	machine learning stanford	飙升
machine learning algorithms	50	machine learning pdf	+250%
data mining	45	online machine learning	+250%
machine learning stanford	45	pattern recognition	+180%
machine learning classification	40	machine learning techniques	+90%

图 5-19 “机器学习”在 Google Trends 中的搜索结果

## 思考题

1. 本章中介绍了用 R 语言进行回归分析的方法。请对下列几组数据进行回归分析。  
(1)  $x=[2,5,8,9,12,15]$ ,  $y=[18,52,78,101,125,148]$   
(2)  $x=[2,5,8,9,12,15]$ ,  $y=[5,120,502,739,1708,3415]$

**提示** 第一组数据适用线性回归，第二组数据适用非线性回归，回归方程为  $y=x \times 20 + \exp(x/5)$ 。

2. 在 Google 关键词统计分析中搜索“机器学习”的中文关键词，描述人们对该关键词兴趣的发展趋势，目前机器学习的哪些分支领域正在成为人们关注的热点。
3. 下载本书例子中的美国地震台数据 earthquakes.csv，分析 2013.3 ~ 2013.4 期间的数据，分震级统计这段时期全球发生的地震，同时做出这段时间的震深与震级散点图，计算震级的累积分布，并显示累积分布图。



## 第 6 章 Chapter 6

# 描述性分析案例

本章将以 R 语言为分析工具对描述性分析案例进行剖析，对于其中涉及的统计分析知识也会做简单介绍，请各位读者按准备篇的指导将 R 语言计算平台搭建好。

## 6.1 数据图形化案例解析

数据是事实，也称观测值，是实验、测量、观察、调查等活动的结果，常以数量的形式给出。数据分析的目的是把隐没在一大批看似杂乱无章的数据中的有用信息集中、萃取和提炼出来，以找出所研究对象的内在规律。

### 6.1.1 点图

下面以 2010 年全国各行业就业调查数据（部分数据如表 6-1 所示，完整数据在本书下载包中）为依据，以点图分析为手段，剖析电子行业劳动报酬水平。

表 6-1 全国各行业就业调查部分数据

行业代码	行业名称	平均劳动报酬	平均教育经费
10000	农林牧渔业	22475	143
10100	农业	17542	230
10300	畜牧业	27958	308
10310	牲畜的饲养	25636	196
20000	采矿业	43713	44
20800	黑色金属矿采选业	43895	42
...	...	...	...



读入如表 6-1 所示的数据文件。代码如下：

```
> read.table("youxiangz.csv",,header=TRUE,sep=",")->jiuye
```

分析行业报酬水平，以电子行业的劳动报酬为例进行讲解，主要步骤如下。

1) 从数据集中筛选电子行业的劳动报酬。代码如下：

```
> jiuye$行业名称[grepl("电子",jiuye$行业名称)]->jyhy
> jiuye$平均劳动报酬[grepl("电子",jiuye$行业名称)]->jygz
> names(jygz)<-jyhy
```

2) 绘制电子行业薪水点图，如图 6-1 所示。从图 6-1 中可清楚地看到 7 个电子行业的薪水分布情况。可将图 6-1 分为若干行，每一行代表一个行业，点在每行的不同位置代表不同的报酬，所有行的数值遵循同一刻度（刻度尺在点图的最下方，从 40000 到 120000 分成 4 个区域）。

3) 找到薪水最高、最低的行业。首先找到图 6-1 中相应行业代表的行，然后在该行找到由点代表的刻度值（点在某行的位置），最后在刻度尺中找到相应数值，读取数值。很明显，代表“电子计算机制造”行业报酬的点在所有行中最贴近右端，属于薪水最高的行业，而“家用电器及电子产品专门零售”行业的数值在最左端，属于电子行业中薪水最低的行业。

```
> dotchart(jygz)
```

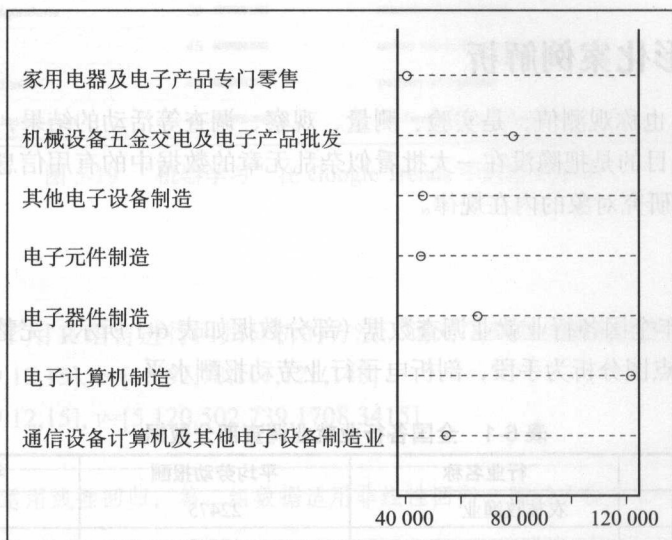


图 6-1 电子行业薪水点图

### 6.1.2 饼图和条形图

下面以饼图和条形图为分析手段，对中介行业的平均劳动报酬进行剖析。

1) 以条形图来表示平均劳动报酬。代码如下:

```
> jiuye$平均劳动报酬[grepl("中介",jiuye$行业名称)]->jygz
> jiuye$行业名称[grepl("中介",jiuye$行业名称)]->jyhy
> names(jygz)<-jyhy
> barplot(jygz,horiz = TRUE)
```

绘制结果如图 6-2 所示。

从图 6-2 可明显观察到,“科技中介服务”行业的平均劳动报酬位居第一,“职业中介服务”位居最后。

2) 除了条形图,还可以用饼图分析平均劳动报酬。代码如下:

```
> pie(jygz)
```

绘制结果如图 6-3 所示。

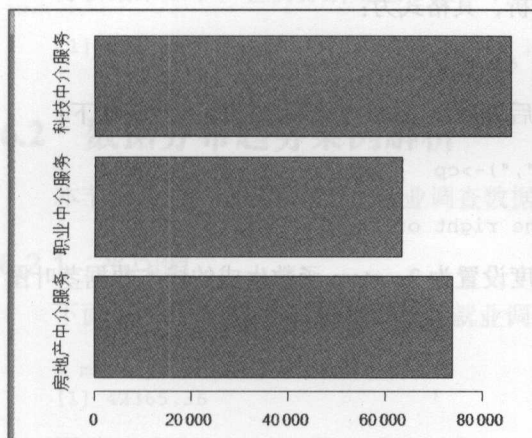


图 6-2 中介行业的平均劳动报酬条形图

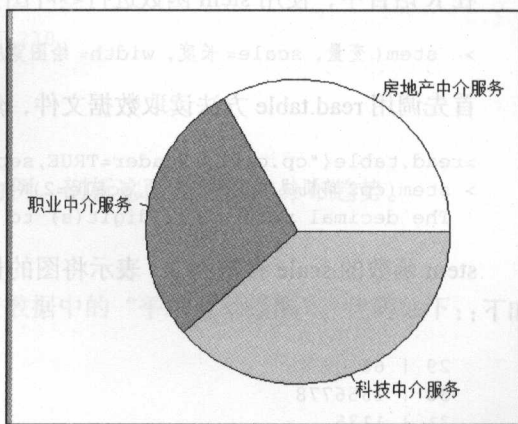


图 6-3 中介行业的平均劳动报酬饼图

观察图 6-3,代表“科技中介服务”的面积所占比重最大,因此属于平均劳动报酬最高的中介服务行业。

### 6.1.3 茎叶图和箱线图

本节将要讲解的茎叶图和箱线图,可能大家平时接触得比较少,但在数据分析中经常用到。茎叶图和箱线图不同于前面的图表,表面看上去比较抽象,一旦掌握了读图的方法后,会发现它们表现数据的能力还是很强的。

#### 1. 茎叶图分析

茎叶图又称“枝叶图”,它的思路是将数组中的数按位数进行比较,然后将数的大小基本不变或变化不大的位作为一个主干,并将变化大的位的数作为分枝,列在主干的后面,这样就可以清楚地看到每个主干后面有几个数,每个数具体是多少。下面以产品单位成本数据为例,分析它的茎叶图,如表 6-2 所示。

表 6-2 产品单位成本数据

序号	产量 (台)	单位成本 (元 / 台)
1	4300	346.23
2	4004	343.34
3	4300	327.46
4	5016	313.27
5	5511	310.75
6	5648	307.61
7	5876	314.56
8	6651	305.72
9	6024	310.82
...	...	...

在 R 语言中，使用 `stem` 函数进行茎叶图分析，其格式为：

```
> stem( 变量, scale= 长度, width= 绘图宽度, atom= 容差)
```

首先调用 `read.table` 方法读取数据文件，然后调用 `stem` 函数绘制茎叶图。代码如下：

```
> read.table("cp.csv",,header=TRUE,sep=",")->cp
> stem(cp$ 单机成本 . 元 . 台 .,scale=2)
The decimal point is 1 digit(s) to the right of the |
```

`stem` 函数的 `scale` 参数为 2，表示将图的长度设置为 2。`stem` 函数生成的成本数据茎叶图如下：

```
29 | 68
30 | 1356778
31 | 1135
32 | 7
33 |
34 | 36
```

茎叶图的每一行表示每个茎与它的叶子，“|”前面是茎，而“|”后面是叶。以最后一行“34|36”为例，“|”前面的“34”是茎，后面的“36”是叶，这行的意思是：百位数为 3，十位数为 4 的数据有两个，分别是：345 和 346。

从茎叶图中可看出，单位成本主要集中在 300 ~ 309 元，因为代表茎 30 的行拥有的叶子最多。此外，茎 33 的行没有一个叶子，这说明没有一件产品的单位成本在 330 ~ 339 元这个范围内。

## 2. 箱线图分析

箱形图提供了一种只用 5 个点来对数据集做简单总结的方式，这 5 个点包括最大值、最小值、中位数、下四分位数和上四分位数。箱形图中最重要的内容是对相关统计点的计算，相关统计点可以通过百分位计算方法进行实现。

以 2010 年全国就业调查数据为例，绘制“平均教育经费”的箱形图并进行分析。R 语言中，实现箱线图分析的相应函数为 `boxplot`。下面的代码绘制平均教育经费的箱形图。

```
> boxplot(jiuye$ 平均教育经费)
```

如图 6-4 所示的箱形图将平均教育经费很形象地分为中心、延伸以及分部状态的全部范围。中间那个箱子的顶部是上四分位数，底部是下四分位数，中间的粗线是中位数位置，箱体由上下伸出的垂直部分表示数据的散布范围。另外在散布范围外还有一些小圆点，那些是异常点，可见平均教育经费有一些特大值，最大的异常值超过了 12 000。

除了箱形图外，在 R 语言中，还可以使用 `fivenum` 函数来分析前面说的 5 个点的概要。代码如下：

```
> fivenum(cp$ 单机成本.元.台.)
```

分析结果如下，它们分别是最小值、下四分位数、中位数、上四分位数、最大值。

```
[1] 296.210 304.275 307.225 313.915 346.230
```

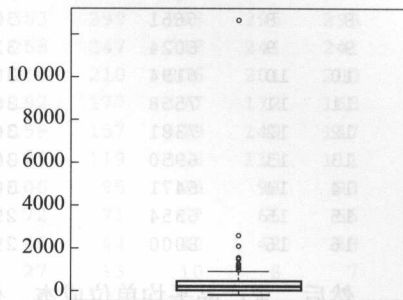


图 6-4 平均教育经费的箱形图

## 6.2 数据分布趋势案例解析

本节继续以产品成本和全国就业调查数据为例，剖析这两类数据的分布趋势。

### 6.2.1 平均值

下面使用 R 语言的 `mean` 函数统计就业调查数据中的“平均劳动报酬”。代码如下：

```
> mean(jiuye[[" 平均劳动报酬 "]])
[1] 42365.36
```

同时，还可以统计一下“平均劳动报酬”和“平均教育经费”。代码如下：

```
> cbind(jiuye[[" 平均劳动报酬 "]], jiuye[[" 平均教育经费 "]])
> apply(jiuye$info, 2, mean)
[1] 42365.365 391.035
```

### 6.2.2 加权平均值

加权平均数与算术平均数类似，但数据集中的每个数据对于平均数的贡献并不是相等的，有些数据要比其他的数据更加重要。因此，在加权平均法中，每个数据都有其相对应的权重。

以产品成本数据为例，剖析加权平均值。首先读取产品成本数据。代码如下：

```
> read.table("cp.csv", , header=TRUE, sep=",") -> cp
> cp
```

	序号	产量.台.	单机成本.元.台.
1	1	4300	346.23
2	2	4004	343.34
3	3	4300	327.46
4	4	5016	313.27
5	5	5511	310.75



6	6	5648	307.61
7	7	5876	314.56
8	8	6651	305.72
9	9	6024	310.82
10	10	6194	306.83
11	11	7558	305.11
12	12	7381	300.71
13	13	6950	306.84
14	14	6471	303.44
15	15	6354	298.03
16	16	8000	296.21

然后,求产品平均单位成本。代码如下:

```
> weighted.mean(cp$ 单机成本 . 元 . 台 . , cp$ 产量 . 台 .)
[1] 309.9866
```

### 6.2.3 数据排序

在 R 语言中,使用 sort 函数进行数据排序。比如,要对“平均教育经费”进行排序,可采用如下代码:

```
> sort(jiuye$ 平均教育经费)
 [1]      0      0      0      2      2      2      6      7      7      8     10     13
[13]     27     30     31     31     31     32     35     37     38     42     42     44
[25]     46     50     51     55     55     62     63     65     66     66     67     71
[37]     72     72     75     75     76     80     89     92     93     93     95     95
[49]    100    100    100    100    100    105    109    110    111    115    118    119
[61]    125    136    138    143    144    145    146    147    147    149    149    157
[73]    159    161    161    162    162    166    168    168    168    172    177    177
[85]    182    184    186    188    190    196    196    196    200    201    206    210
[97]    210    212    212    221    224    225    230    230    241    241    247    247
[109]    258    260    267    267    276    276    277    282    295    295    298    299
[121]    303    304    305    306    308    314    315    317    330    332    337    340
[133]    341    342    348    367    369    371    374    374    389    389    396    402
[145]    405    409    416    422    422    423    431    436    443    454    455    461
[157]    466    470    486    502    522    524    535    551    551    554    555    557
[169]    563    571    582    645    679    682    692    722    738    753    768    782
[181]    818    830    832    840    840    858    890    890    890    986    995    1096
[193]   1131   1198   1255   1469   1553   2087   2564  12645
```

排序后,可以初步发现,这些行业的教育经费中,最大的有 12645,而最小的除 0 之外还有 2,不同行业之间的教育经费差异很大。

也可以改变排序顺序,通过指定 decreasing 参数为 TRUE,实现按从大到小的顺序排列。代码如下:

```
> sort(jiuye$ 平均教育经费 , decreasing=TRUE)
 [1] 12645  2564  2087  1553  1469  1255  1198  1131  1096  995  986  890
[13]   890   890   858   840   840   832   830   818   782   768   753   738
[25]   722   692   682   679   645   582   571   563   557   555   554   551
[37]   551   535   524   522   502   486   470   466   461   455   454   443
```



```

[49] 436 431 423 422 422 416 409 405 402 396 389 389
[61] 374 374 371 369 367 348 342 341 340 337 332 330
[73] 317 315 314 308 306 305 304 303 299 298 295 295
[85] 282 277 276 276 267 267 260 258 247 247 241 241
[97] 230 230 225 224 221 212 212 210 210 206 201 200
[109] 196 196 196 190 188 186 184 182 177 177 172 168
[121] 168 168 166 162 162 161 161 159 157 149 149 147
[133] 147 146 145 144 143 138 136 125 119 118 115 111
[145] 110 109 105 100 100 100 100 100 95 95 93 93
[157] 92 89 80 76 75 75 72 72 71 67 66 66
[169] 65 63 62 55 55 51 50 46 44 42 42 38
[181] 37 35 32 31 31 31 30 27 13 10 8 7
[193] 7 6 2 2 2 0 0 0

```

## 6.2.4 中位数

中位数比平均值更有稳健性，因为它不受偏态分布的影响。代码如下：

```

> median(jiuye$ 平均教育经费) # 中位数
[1] 222.5
> mean(jiuye$ 平均教育经费) # 平均数
[1] 391.035

```

教育经费的中位数 222.5 与它的平均值 391.035 有一定差距，这说明平均教育经费不是对称分布的。

## 6.2.5 极差、半极差

极差是一组数据中最大数据与最小数据的差，用来刻画一组数据的离散程度，反映变量分布的变异范围和离散幅度，在样本总体中任何两个单位的标准值之差都不能超过极差。同时，它还能体现一组数据波动的范围。下面的代码计算“平均教育经费”的极差。

```

> max(jiuye$ 平均教育经费) - min(jiuye$ 平均教育经费)
[1] 12645

```

相对极差而言，四分位数间距（上四分位数与下四分位数之差）更稳定，它不受两端个别极大值或极小值的影响，可理解为中间 50% 观察值的极差，因此又被称为半极差。

四分位数是统计学中分位数的一种，即把所有数值由小到大排列并分成 4 等份，处于 3 个分隔点位置的得分就是四分位数，下四分位数是所有数据由小到大排列后处于 25% 位置的数，上四分位数是所有数据由小到大排列后处于 75% 位置的数。四分位数在 R 语言中用 `quantile` 函数求解，下面的代码计算了“平均教育经费”和“平均劳动报酬”的四分位数。

```

> quantile(jiuye$ 平均教育经费)
 0%    25%    50%    75%   100%
0.0  100.0  222.5  425.0 12645.0
> quantile(jiuye$ 平均劳动报酬)
 0%    25%    50%    75%   100%
13624.0 28607.5 37681.0 51762.0 150098.0

```

变异度反映数据围绕中心位的离散度,四分位数间距数值越大,变异度越大,反之,变异度越小。在 R 语言中使用 IQR 函数求解四分位数间距,下面的代码计算“平均教育经费”和“平均劳动报酬”的四分位数间距。

```
> IQR(jiuye$ 平均教育经费)
[1] 325
> IQR(jiuye$ 平均劳动报酬)
[1] 23154.5
```

从执行结果来看,“平均教育经费”相比“平均劳动报酬”变异度小很多。

## 6.2.6 方差

方差是重要的数据分散程度度量指标。其计算公式为:

$$S^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

在 R 语言中,可使用 var 函数统计方差。下面的代码计算“平均教育经费”的方差。

```
> var(jiuye$ 平均教育经费)
[1] 883263.6
```

## 6.2.7 标准差

标准差也是重要的数据分散程度度量指标。其计算公式为:

$$S = \sqrt{S^2} = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

在 R 语言中使用 sd 函数统计标准差。下面的代码计算“平均教育经费”的标准差。

```
> sd(jiuye$ 平均教育经费)
[1] 939.821
```

## 6.2.8 变异系数、样本平方和

### 1. 变异系数

变异系数,又称“离散系数”,是概率分布离散程度的一个归一化量度,其定义为标准差与平均值之比。变异系数的计算公式为:

$$C.V. = \frac{S}{\bar{x}} \times 100\%$$

上式中  $S$  表示标准差,  $\bar{x}$  表示平均值。变异系数越小,变异程度越小;反之,变异系数越大,变异程度越大。下面的代码计算了“平均教育经费”的变异系数。

```
> sd(jiuye$ 平均教育经费) / mean(jiuye$ 平均教育经费)
[1] 2.403419
```

再看看“平均劳动报酬”的变异系数。代码如下:

```
> sd(jiuye$ 平均劳动报酬) / mean(jiuye$ 平均劳动报酬)
```

```
[1] 0.4916487
```

可见,“平均教育经费”相对于“平均劳动报酬”分布更分散,因为它的变异系数更高。

## 2. 样本平方和

样本校正平方和 (CSS) 为样本与均值差的平方求和。下面的代码计算“平均教育经费”的样本校正平方和。

```
> sum((jiuye$ 平均教育经费 - mean(jiuye$ 平均教育经费))^2)
[1] 175769451
```

样本未校正平方和 (USS) 为样本值平方的求和。下面的代码计算了“平均教育经费”的样本未校正平方和。

```
> sum(jiuye$ 平均教育经费^2)
[1] 206351125
```

## 6.2.9 偏度系数、峰度系数

### 1. 偏度系数

在统计学中,偏度系数是用于衡量实数随机变量概率分布的不对称性的。偏度的值可以为正,可以为负,是无量纲的量,其取值通常为  $-3 \sim +3$ ,其绝对值越大,表明偏斜程度越大。均值右侧更分散的数据偏度系数为正,左侧更分散的数据偏度系数为负。

偏度系数的计算公式为:

$$\gamma = \frac{n}{(n-1)(n-2)S^3} \sum (x_i - \bar{x})^3$$

在 R 语言中,可用如下代码计算“平均教育经费”的偏度系数。

```
> mean(jiuye$ 平均教育经费) -> mymean
> sd(jiuye$ 平均教育经费) -> mysd
> length(jiuye$ 平均教育经费) -> myn
> jiuye$ 平均教育经费 -> x
> myn / ((myn-1) * (myn-2)) * sum((x - mymean)^3) / mysd^3
[1] 11.36649
```

### 2. 峰度系数

峰度系数衡量实数随机变量概率分布的峰态,峰度高就意味着方差增大是由低频度的大于或小于平均值的极端差值引起的。当数据分布为正态分布时,峰度系数近似为 0; 当数据分布较正态分布的尾部更分散时,峰度系数为正,两侧的极端数据较多; 除此以外,峰度系数为负,两侧的极端数据较少。

峰度系数计算公式为:

$$K = \frac{n(n+1)}{(n-1)(n-2)(n-3)S^4} \sum (x_i - \bar{x})^4 - \frac{3(n-1)^2}{(n-2)(n-3)}$$

在 R 语言中,可用如下代码计算“平均教育经费”的峰度系数。

```
> mean(jiuye$ 平均教育经费) -> mymean
```

```

> sd(jiuye$ 平均教育经费 )->mysd
> length(jiuye$ 平均教育经费 )->myn
> jiuye$ 平均教育经费 ->x
> ((myn*(myn+1))/((myn-1)*(myn-2)*(myn-3))*sum((x-mymean)^4)/mysd^4-
(3*(myn-1)^2)/((myn-2)*(myn-3)))
[1] 146.8809

```

## 6.3 正态分布案例解析

### 6.3.1 正态分布函数

对于一维实随机变量  $X$ , 设它的累积分布函数是  $F_X(x)$ 。如果存在可测函数  $f_X(x)$ , 满足:

$$\forall -\infty < a < \infty, F_X(a) = \int_{-\infty}^a f_X(x) dx$$

那么  $X$  是一个连续型随机变量, 并且  $f_X(x)$  是它的概率密度函数。

累积分布函数, 又叫累计分布函数, 是概率密度函数的积分, 能完整地描述一个实随机变量  $X$  的概率分布情况。对于所有实数  $x$ , 累积分布函数的定义如下:

$$F(x) = P(X \leq x)$$

正态分布的累积分布函数为:

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x \exp\left\{-\frac{(t-\mu)^2}{2\sigma^2}\right\} dt$$

其中,  $\mu$  是均值,  $\sigma$  是方差。

正态分布的概率密度曲线通常如图 6-5 所示。

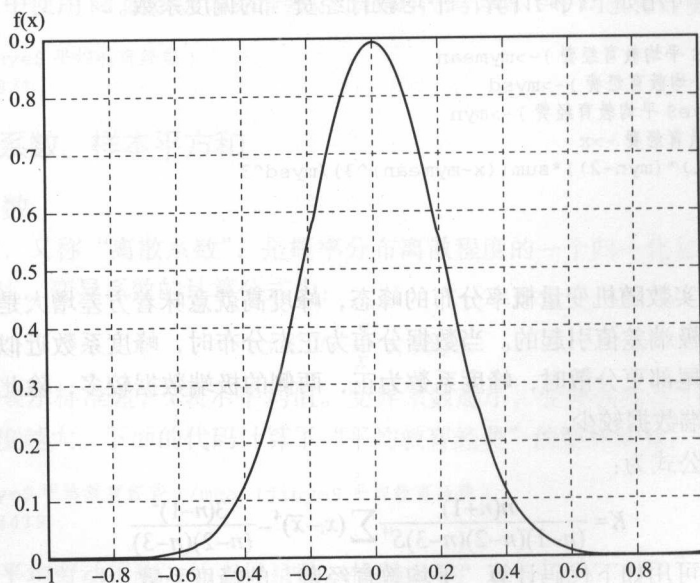


图 6-5 正态分布的概率密度曲线

说到正态分布,不得不提一下偏态分布,偏态分布是指频数分布不对称,集中位置偏向于一侧,若集中位置偏向数值小的一侧,则称为正偏态分布;如果集中位置偏向数值大的一侧,则称为负偏态分布。如图6-6所示,左边为负偏态,右边为正偏态。

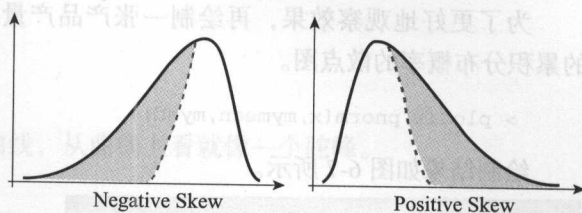


图 6-6 偏态分布

### 6.3.2 峰度系数分析

可用峰度系数计算“平均劳动报酬”

相对于“平均教育经费”哪个更接近正态分布。代码如下:

```
> mean(jiuye$ 平均劳动报酬)->mymean
> sd(jiuye$ 平均劳动报酬)->mysd
> length(jiuye$ 平均劳动报酬)->myn
> jiuye$ 平均劳动报酬 ->x
> ((myn*(myn+1))/((myn-1)*(myn-2)*(myn-3))*sum((x-mymean)^4)/mysd^4-
(3*(myn-1)^2)/((myn-2)*(myn-3)))
[1] 5.417817
```

上面计算出了“平均劳动报酬”的峰度系数为 5.417 817,“平均教育经费”的峰度系数为 146.8809 (见 6.2.9 节)。这两个峰度系数表明,“平均劳动报酬”相对于“平均教育经费”更接近正态分布。

从下面的分析中可以发现,产品产量最适合正态分布模型,因为它的峰度系数仅为 -0.683 072 8,非常接近正态分布。

```
> mean(cp$ 产量.台.)->mymean
> sd(cp$ 产量.台.)->mysd
> length(cp$ 产量.台.)->myn
> cp$ 产量.台.->x
> ((myn*(myn+1))/((myn-1)*(myn-2)*(myn-3))*sum((x-mymean)^4)/mysd^4-
(3*(myn-1)^2)/((myn-2)*(myn-3)))
[1] -0.6830728
```

### 6.3.3 累积分布概率

在使用 pnorm 求产品产量的分布函数时,对应的每个实数随机变量都有其累积分布概率。下面的代码计算产品产量的累积分布概率。

```
> mean(cp$ 产量.台.)->mymean
> sd(cp$ 产量.台.)->mysd
> length(cp$ 产量.台.)->myn
> cp$ 产量.台.->x
> x
[1] 4300 4004 4300 5016 5511 5648 5876 6651 6024 6194 7558 7381 6950 6471
[15] 6354 8000
> pnorm(x,mymean,mysd)
[1] 0.07435941 0.04519643 0.07435941 0.20013522 0.33567136 0.37868351
[7] 0.45345196 0.70390728 0.50306546 0.55994848 0.90310411 0.87500925
```



[13] 0.78449233 0.64954647 0.61239714 0.95270286

为了更好地观察效果,再绘制一张产品产量的累积分布概率的散点图。

```
> plot(x,pnorm(x,mymean,mysd))
```

绘制结果如图 6-7 所示。

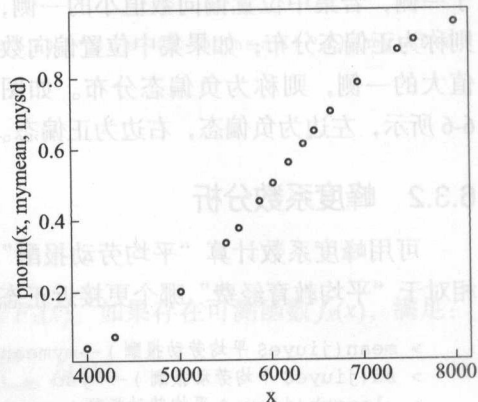


图 6-7 产品产量累积分布

## 6.3.4 概率密度函数

### 1. 概率密度概述

一个连续型随机变量的概率密度函数(简称为密度函数)是描述这个随机变量的输出值在某个确定的取值点附近的可能性的函数。随机变量的取值落在某个区域之内的概率则是概率密度函数在这个区域上的积分,当概率密度函数存在的时候,累积分布函数则是概率密度函数的积分。

正态分布的概率密度函数为:

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{(t-\mu)^2}{2\sigma^2}\right\}$$

其中,  $\mu$  是均值,  $\sigma$  是方差。

其概率密度曲线关于  $x=\mu$  对称。

### 2. 概率密度函数计算

在 R 语言中,使用 `dnorm` (变量,平均值,标准差)求解正态分布概率密度函数。下面的代码计算产品产量的概率密度。

```
> mean(cp$产量.台.)->mymean
> sd(cp$产量.台.)->mysd
> length(cp$产量.台.)->myn
> cp$产量.台.->x
> dnorm(x,mymean,mysd)
[1] 1.184240e-04 8.009886e-05 1.184240e-04 2.358477e-04 3.070239e-04
[6] 3.202882e-04 3.336542e-04 2.910430e-04 3.359337e-04 3.321435e-04
[11] 1.444115e-04 1.733374e-04 2.463862e-04 3.120546e-04 3.225207e-04
[16] 8.307503e-05
```

查看产品产量均值,得到如下结果:

```
> mymean
[1] 6014.875
```

绘制散点图如图 6-8 所示。可以看到该曲线接近于以 6014.875 为对称点的对称分布。绘制代码如下:

```
>plot(x,dnorm(x,mymean,mysd))
```

此外, `rnorm` 还可以返回正态分布随机数, 调用格式为 `rnorm(长度, 平均值, 标准差)`。比如:

```
>rnorm(50,0,1)->rx
> plot(rx,dnorm(rx))
```

如图 6-9 所示是一个经典的正态密度曲线, 从曲线上看就像一个驼峰。

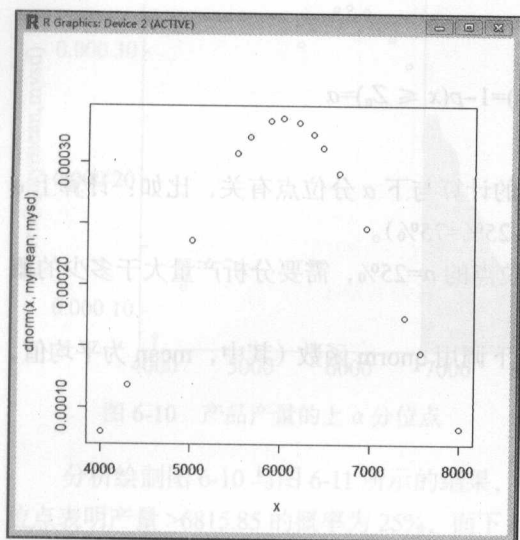


图 6-8 产品产量概率密度图

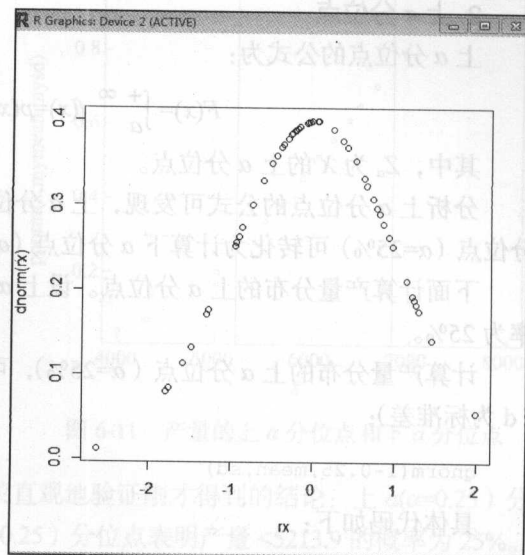


图 6-9 正态密度曲线

### 6.3.5 分位点

分位点分为上  $\alpha$  分位点与下  $\alpha$  分位点。

#### 1. 下 $\alpha$ 分位点

可从概率密度函数的角度理解下  $\alpha$  分位点。设连续随机变量  $X$  的累积分布函数为  $F(x)$ , 密度函数为  $f(x)$ , 则有:

$$F(x) = \int_{-\infty}^x f(x) = P(X \leq Z_\alpha) = \alpha$$

上式的含义为: 连续随机变量  $X$  小于等于  $Z_\alpha$  的概率为  $\alpha$ 。在这里, 称  $Z_\alpha$  是  $X$  的下  $\alpha$  分位点。

下面计算产量分布的下  $\alpha$  分位点。设下  $\alpha$  分位点的  $\alpha=25\%=0.25$ , 需要分析产量小于多少的概率为 25%。

计算产量分布的下  $\alpha$  分位点 ( $\alpha=25\%$ ), 可如下调用 `qnorm` 函数 (其中, `mean` 为平均值, `sd` 为标准差):

```
qnorm(0.25, mean, sd)
```

具体代码如下:

```
> mean(cp$产量.台.)->mymean
> sd(cp$产量.台.)->mysd
> qnorm(0.25,mean=mymean,sd=mysd)
[1] 5213.9
```

上面的计算结果表明,产量 < 5213.9 的概率为 25%。

## 2. 上 $\alpha$ 分位点

上  $\alpha$  分位点的公式为:

$$F(x) = \int_a^{+\infty} f(x) = p(x > Z_\alpha) = 1 - p(x \leq Z_\alpha) = \alpha$$

其中,  $Z_\alpha$  为  $X$  的上  $\alpha$  分位点。

分析上  $\alpha$  分位点的公式可发现,上  $\alpha$  分位点的计算与下  $\alpha$  分位点有关,比如:计算上  $\alpha$  分位点 ( $\alpha=25\%$ ) 可转化为计算下  $\alpha$  分位点 ( $\alpha=1-25\%=75\%$ )。

下面计算产量分布的上  $\alpha$  分位点。设上  $\alpha$  分位点的  $\alpha=25\%$ ,需要分析产量大于多少的概率为 25%。

计算产量分布的上  $\alpha$  分位点 ( $\alpha=25\%$ ),可如下调用 qnorm 函数 (其中, mean 为平均值, sd 为标准差):

```
qnorm(1-0.25,mean,sd)
```

具体代码如下:

```
> mean(cp$产量.台.)->mymean
> sd(cp$产量.台.)->mysd
> qnorm(0.75,mean=mymean,sd=mysd)
[1] 6815.85
```

上面的计算结果表明,产量 > 6815.85 的概率为 25%,即上  $\alpha$  分位点的公式中,  $Z_\alpha$  为产量 6815.85,上  $\alpha$  分位点的  $\alpha$  为 0.25,可用下式表示:

$$P(x > Z_\alpha) = p(x > 6815.85) = 0.25$$

## 3. 绘效果图

1) 通过下面 R 语句绘制如图 6-10 所示的产品产量的概率密度图,  $P(x > 0.25)$  为图 6-10 中的阴影面积 (根据积分的几何意义,累积分布函数  $F(x)$  是密度函数  $f(x)$  的积分,图中若干点组成了密度函数曲线,而曲线与  $X$  轴围成的面积则为累积分布)。

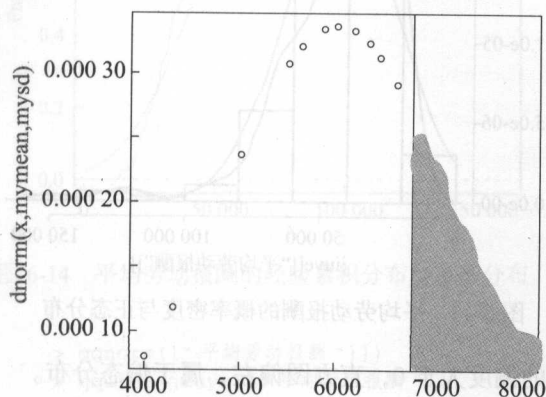
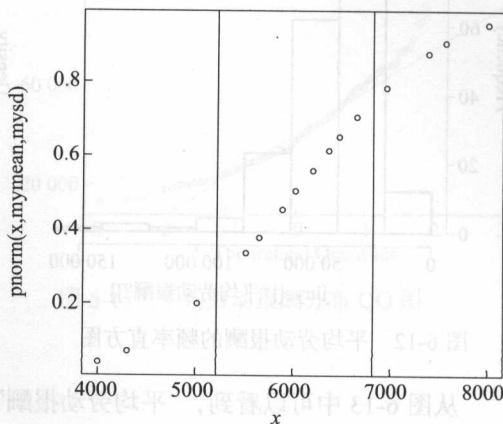
```
> mean(cp$产量.台.)->mymean
> sd(cp$产量.台.)->mysd
> cp$产量.台.->x
> plot(x,dnorm(x,mymean,mysd))
> abline(v=6815.85)
```

2) 绘制产品产量的累积分布图 (如图 6-11 所示)。可绘制一个产品产量的上  $\alpha(\alpha=0.25)$  分位点和下  $\alpha(\alpha=0.25)$  分位点的效果图。绘制代码如下:

```

> mean(cp$产量.台.)->mymean
> sd(cp$产量.台.)->mysd
> cp$产量.台.->x
> plot(x,pnorm(x,mymean,mysd))
> abline(v=6815.85)
> abline(v=5213.85)

```

图 6-10 产品产量的上  $\alpha$  分位点图 6-11 产量的上  $\alpha$  分位点和下  $\alpha$  分位点

分析绘制图 6-10 与图 6-11 所示的结果,能较直观地验证刚才得到的结论:上  $\alpha(\alpha=0.25)$  分位点表明产量  $>6815.85$  的概率为 25%,而下  $\alpha(\alpha=0.25)$  分位点表明产量  $<5213.9$  的概率为 25%。

### 6.3.6 频率直方图

在 R 语言中,可使用 hist 语句(设 freq 参数为 TRUE)生成频率直方图。下面的代码绘制“平均劳动报酬”的频率直方图。

```
> hist(jiuye[["平均劳动报酬"]],freq=TRUE)
```

绘制结果如图 6-12 所示。

### 6.3.7 核概率密度与正态概率分布图

#### 1. 核概率密度与正态概率

下面考虑让“平均劳动报酬”的概率密度与正态分布在一张图中显示出来,这样就能更好地看清数据的分布情况。示例代码如下:

```

> hist(jiuye[["平均劳动报酬"]],freq=FALSE)
> lines(density(jiuye[["平均劳动报酬"]]),col="red")
> x<-c(0:ceiling(max(jiuye[["平均劳动报酬"]]))))
> lines(x,dnorm(x,mean(jiuye[["平均劳动报酬"]]),sd(jiuye[["平均劳动报酬"]])),col="blue")

```

绘制结果如图 6-13 所示。

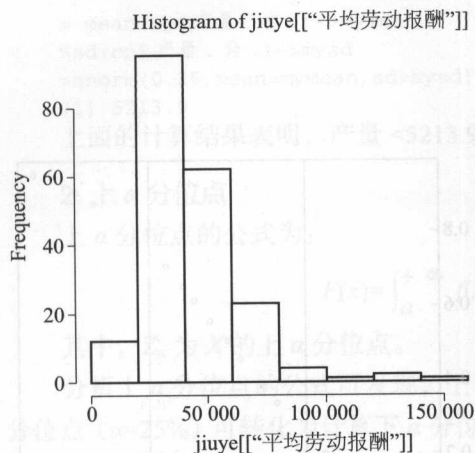


图 6-12 平均劳动报酬的频率直方图

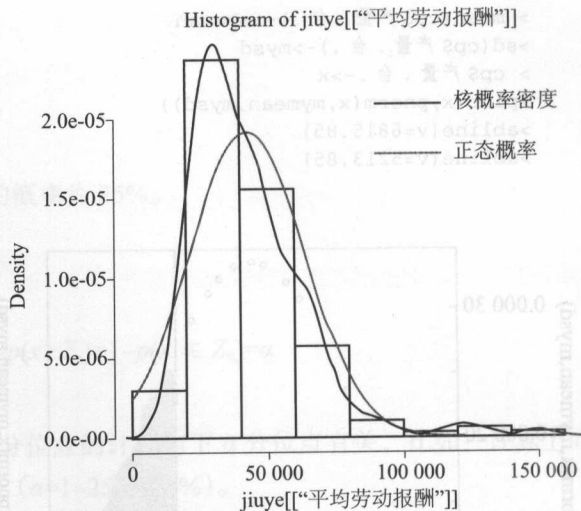


图 6-13 平均劳动报酬的概率密度与正态分布

从图 6-13 中可以看到,“平均劳动报酬”的偏度大于 0,直方图偏左,属于偏态分布。

## 2. 经验累积分布与正态分布

经验分布函数是指根据样本构造的概率分布函数。设  $x_1, x_2, \dots, x_n$  为一组样本,定义函数  $m(x)$  表示样本中小于或者等于  $x$  的样本个数,则称函数

$$F_n^x = \frac{m(x)}{n}$$

为样本  $x_1, x_2, \dots, x_n$  的经验分布函数。

下面的代码绘制“平均劳动报酬”的经验累积分布与正态分布。

```
> plot(ecdf(jiuye[\"平均劳动报酬\"]),verticals=TRUE,do.p=FALSE)
> lines(x,pnorm(x,mean(jiuye[\"平均劳动报酬\"]),sd(jiuye[\"平均劳动报酬\"]))),col=\"blue\")
```

绘制结果如图 6-14 所示。其中,光滑的线为累积正态分布曲线,不光滑的线为经验累积分布曲线。

## 6.3.8 正态检验与分布拟合

### 1. QQ 图

QQ 图可以测试数据分布是否近似为某种类型分布。如果近似于正态分布,则数据点接近下面方程表示的直线。

$$y = \sigma x + \mu$$

其中,  $\sigma$  为标准差,  $\mu$  为平均数。

比如,可用它来测试“平均劳动报酬”分布是否近似于正态分布。在 R 语言中,使用 `qqnorm` 函数来画数据点图,使用 `qqline` 函数画这根直线,如图 6-15 所示。代码如下:



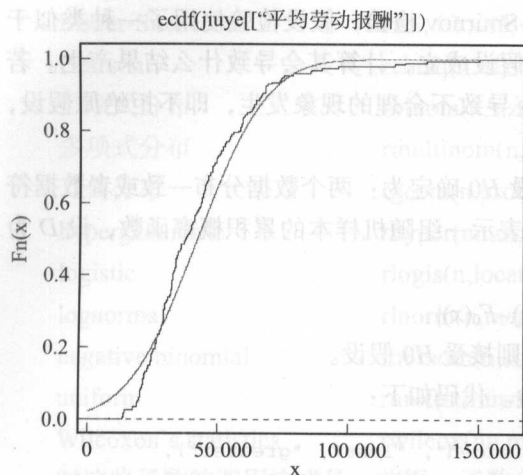


图 6-14 平均劳动报酬的经验累积分布与正态分布

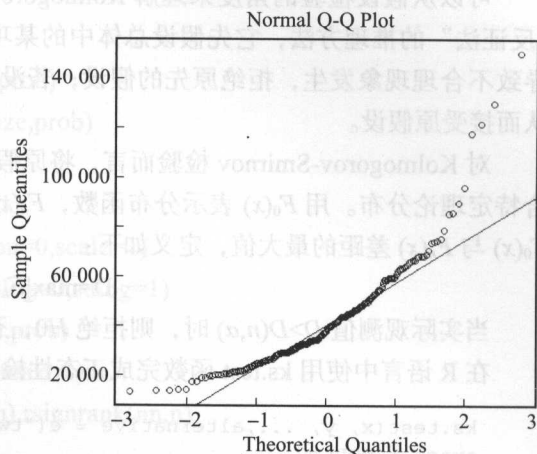


图 6-15 平均劳动报酬分布 QQ 图

```
> qqnorm(["平均劳动报酬"])
> qqline(jiuye(["平均劳动报酬"]))
```

从图 6-15 来看, 平均劳动报酬离标准正态分布还是有差距的。

相对于“平均劳动报酬”, 产品产量就非常接近正态分布。代码如下:

```
> qqnorm(cp$产量.台.)
> qqline(cp$产量.台.)
```

绘制出的 QQ 图如图 6-16 所示。

## 2. 正态检验与分布拟合

1) W 检验。W 检验可以检验数据是否符合正态分布。在 R 语言中使用函数 `shapiro.test()` 进行正态 W 检验。代码如下:

```
> shapiro.test(cp$产量.台.)
Shapiro-Wilk normality test
data: cp$产量.台.
W = 0.9671, p-value = 0.7903
```

上述代码中出现了  $p$  值, 其作用是: 当  $p$  值小于某个显著水平  $\alpha$  (如 0.05) 时, 认为样本不是来自于正态分布的总体。此例中,  $0.7903 > 0.05$ , 可认为产量是正态分布的。

2) Kolmogorov-Smirnov 检验。Kolmogorov-Smirnov 检验用于检验单一样本是否来自某一特定分布, 比如检验一组数据是否为正态分布。它的检验方法是以样本数据的累计频数分布与特定理论分布做比较, 若两者间的差距很小, 则该样本取自某特定分布族。可比较一个频率分布  $f(x)$  与理论分布  $g(x)$ , 或者两个观测值分布来完成检验。

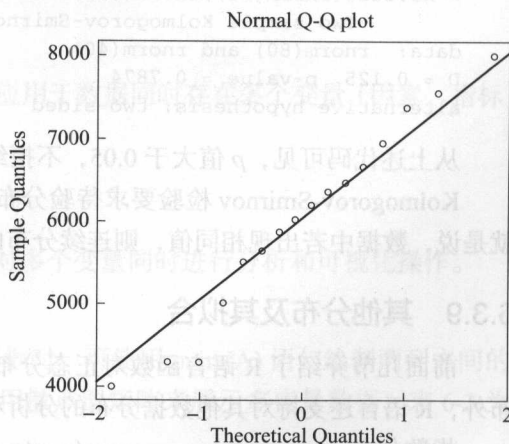


图 6-16 产品产量 QQ 图

可以从假设检验的角度来理解 Kolmogorov-Smirnov 检验, 假设检验使用了一种类似于“反证法”的推理方法, 它先假设总体中的某项假设成立, 计算其会导致什么结果产生。若导致不合理现象发生, 拒绝原先的假设; 若没有导致不合理的现象发生, 即不拒绝原假设, 从而接受原假设。

对 Kolmogorov-Smirnov 检验而言, 将原假设  $H_0$  确定为: 两个数据分布一致或者数据符合特定理论分布。用  $F_0(x)$  表示分布函数,  $F_n(x)$  表示一组随机样本的累积概率函数, 设  $D$  为  $F_0(x)$  与  $F_n(x)$  差距的最大值, 定义如下:

$$D = \max |F_n(x) - F_0(x)|$$

当实际观测值  $D > D(n, \alpha)$  时, 则拒绝  $H_0$ , 否则接受  $H_0$  假设。

在 R 语言中使用 `ks.test` 函数完成正态性检验。代码如下:

```
ks.test(x, y, ..., alternative = c("two.sided", "less", "greater"),
exact = NULL)
```

上述代码中有 4 个参数, 第一个参数  $x$  为观测值向量; 第二个参数  $y$  为第二观测值向量或者累计分布函数, 或者一个真正的累积分布函数, 如 `pnorm`, 只对连续 CDF 有效; 第三个参数指明是单侧检验还是双侧检验; `exact` 参数为 `NULL` 或者一个逻辑值, 表明是否需要计算精确的  $P$  值。比如: 要生成两个随机的正态分布, 然后检验这两个分布是否是同一类型的分类。其示例代码如下:

```
> ks.test(rnorm(80), rnorm(40))
Two-sample Kolmogorov-Smirnov test
data: rnorm(80) and rnorm(40)
D = 0.125, p-value = 0.7874
alternative hypothesis: two-sided
```

从上述代码可见,  $p$  值大于 0.05, 不拒绝原假设, 因此可认为这两个分布是同一类型。

Kolmogorov-Smirnov 检验要求待验分布是连续的, 连续分布出现相同值的概率为 0, 也就是说, 数据中若出现相同值, 则连续分布的假设不成立。

### 6.3.9 其他分布及其拟合

前面几节介绍了 R 语言函数对正态分布的支持 (函数名除前缀外为 `norm`)。除了正态分布外, R 语言还支持对其他数据分布的分析和拟合, 如下所示:

指数分布	<code>rexp(n, rate=1)</code>
gamma 分布	<code>rgamma(n, shape, scale=1)</code>
泊松分布	<code>rpois(n, lambda)</code>
Weibull 分布	<code>rweibull(n, shape, scale=1)</code>
Cauchy 分布	<code>rcauchy(n, location=0, scale=1)</code>
beta 分布	<code>rbeta(n, shape1, shape2)</code>
S(tudent) 分布	<code>rt(n, df)</code>

Fisher-Snedecor	rf(n,df1,df2)
Pearson	rchisq(n,df)
二项式分布	rbinom(n,size,prob)
多项式分布	rmultinom(n,size,prob)
几何分布	rgeom(n,prob)
hypergeometric	rhyper(nn,m,n,k)
logistic	rlogis(n,location=0,scale=1)
lognormal	rlnorm(n,meanlog=0,sdlog=1)
negative binomial	rnbinom(n,size,prob)
uniform	runif(n,min=0,max=1)
Wilcoxon's statistics	rwilcox(nn,m,n),rsignrank(nn,n)

对这些函数的调用格式是：前缀 + 函数名。

前缀规则如下：

密度函数：d

累计概率分布函数：p

分布函数的反函数：q

相同分布的随机数：r

## 6.4 多变量分析

多变量分析是统计分析方法中的一种，通常应用于数据同时存在多个变量（因素、指标）的场景，是对单变量统计分析的发展。

### 6.4.1 多变量数据分析

多变量数据分析也称为多元数据分析，是指对多个变量同时进行分析和可视化操作。

#### 1. 求职情况散点图矩阵

在 R 语言中，当 A 是一个数值型矩阵或数据框时，可使用 `pairs(A)` 语句绘制两列之间的散点图矩阵。下面以求职情况为例，讲解如何使用散点图矩阵来展示多变量数据，表 6-3 为第二季度市场求职指数情况表。

表 6-3 第二季度市场求职指数情况表

年 度	求职人数	绝对求职指数	相对求职指数
2008	3 045 412	100	100
2009	3 413 202	112	112
2010	3 902 961	128	121
2011	3 675 531	121	106

(续)

年 度	求职人数	绝对求职指数	相对求职指数
2012	3 765 853	124	107
2013	3 562 515	117	100
2014	3 350 834	110	94

输入以下 R 代码对表 6-3 的求职数据进行分析:

```
> ejdqz<-read.csv("ejdqz.csv")
> ejdqz
  年度 求职人数 绝对求职指数 相对求职指数
1 2008 年 3045412      100      100
2 2009 年 3413202      112      112
3 2010 年 3902961      128      121
4 2011 年 3675531      121      106
5 2012 年 3765853      124      107
6 2013 年 3562515      117      100
7 2014 年 3350834      110      94
> pairs(ejdqz)# 绘制散点图矩阵, 结果如图 6-17 所示
```

表 6-3 中求职数据的散点图矩阵如图 6-17 所示。

查看散点图矩阵有以下两种方式:

1) 首先, 固定代表某一变量的某一行 (或代表某些变量的某几列), 然后查看与代表其他某变量的某一行 (或代表其他变量的某几行) 交叉处的图。

2) 首先, 固定代表某一变量的某一行 (或代表某些变量的某几行), 然后查看与代表其他某变量的某一行 (或代表其他变量的某几列) 交叉处的图。

例如, 可以定位于年度, 假设需要查看图 6-17 所示的 2012 年的数据, 就将目光定位于第一行, 求职人数接近 380 万, 而绝对求职指数略高于 120, 相对求职指数在 105 到 110 之间。又比如假设需要查看求职人数在 340 万以上, 绝对求职指数在 115 以上的年份, 这样就定位于求职人数和年份, 先锁定于第 1 列和第 3 列, 在第 2 列 (求职人数) 的第 1 行找到 340 万以上 (刻度从左到右数的第 2 个位置以右) 的部分, 将该部分设为 A 区, 第 3 列 (绝对求职指数) 的第 1 行找到 115 万以上 (下

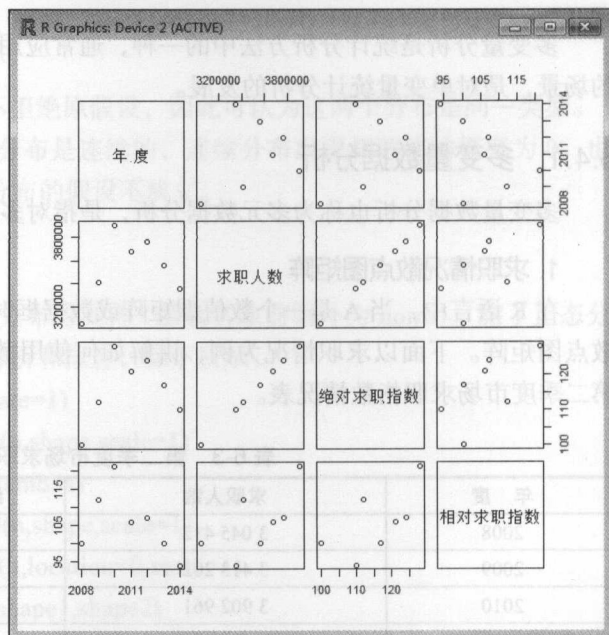


图 6-17 求职散点图矩阵



面最后一行绝对求职指数列对应的刻度表上,从左到右数第4个位置以右)的部分,将该部分设为B区,在A区和B区分别可以找到3个点,A区和B区的3个点互相一一对应,这3个点都属于A、B两个区的共同点,查找这3个点对应的年份(右上角的年份刻度表)为2010、2011、2012。如果A区和B区仅能找到两个共同点,那么就查找这两个共同点所对应的年份。

分析图 6-17 可以看出,2008 年到 2010 年期间的求职指数在上升,市场求职人数出现了扩张态势,在经历 2011 年到 2012 年的小波动后,从 2012 年开始相对求职指数在逐年下降,市场求职人数呈现收缩态势。

## 2. 学生成绩散点图矩阵

首先,输入如下的 R 代码加载并显示学生成绩数据:

```
> source<-read.csv("xscj.csv")
> source
```

	学号	期末考试	平时成绩	性别
1	201	60	74	1
2	202	66	64	1
3	203	91	82	0
4	204	94	49	0
5	205	60	88	0
6	206	48	48	0
7	207	74	84	0
8	208	45	35	0
9	209	97	89	1
10	210	74	98	0
11	211	67	64	0
12	212	50	50	0
13	213	85	94	1
14	214	70	64	0
.....				
.....				

然后,绘制散点图矩阵,代码如下:

```
> pairs(source)#如图 6-18 所示,
性别为 1 则表示男,性别为 0 则表示女。
```

学生成绩散点图矩阵如图 6-18 所示。

从图 6-18 可以看出,第 3 行第 2 列的散点图表明:平时成绩与期末考试成绩有很大的线性相关性,平时成绩较好的,期末考试成绩也较好;平时成绩较差的,期末考试成绩普遍也较差。观察第 3 行第 4 列的散点图,可以看到男生(性别为 1)的成绩分布比较均匀,范围大致为 40 分到 95 分,

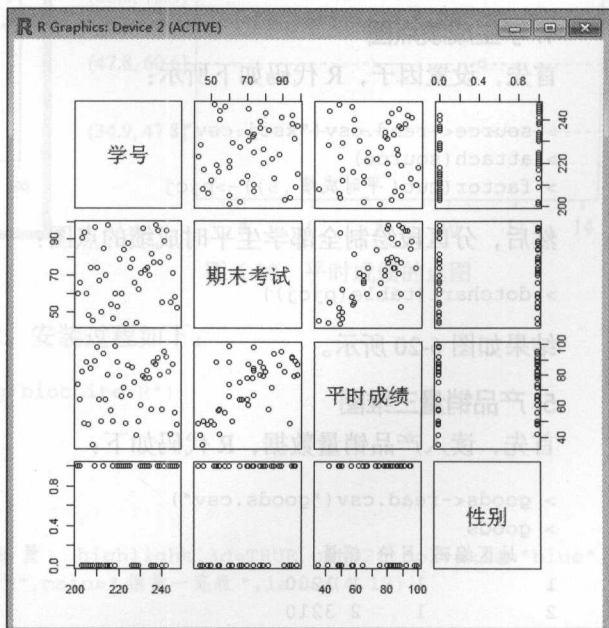


图 6-18 学生成绩散点图矩阵



而女生(性别为0)的成绩分布则集中在两部分,第一部分为60分以下不及格的,这部分的 比例相对于第二部分略小,第二部分集中在80分以上,这部分的 比例较大。

### 3. 学生成绩协同图

协同图(coplot)是一种多变量的探索性分析图形,R语句的基本形式为coplot(y~x|z), 其中x和y是数值型向量,z是同长度的因子,对于z的每一水平,均绘制相应组的x和y 的散点图。下面按性别分组来绘制学生成绩的协同图。

首先,设置因子,R代码如下:

```
> source<-read.csv("xscj.csv")
> attach(source)
> factor(性别,labels = c("女","男"))->sex
> sex
[1] 男 男 女 女 女 女 女 女 男 女 女 女 男 女 女 女 男 男 男 男 男 男
[25] 男 女 女 女 女 男 男 男 男 男 女 女 女 男 男 女 男 女 男 男 男 男
Levels: 女 男
```

然后,以性别来分组,绘制图形,R代码如下:

```
> coplot(平时成绩~期末考试|sex)
```

结果如图6-19所示。

从图6-19可以看出,男生的平时成绩与期末考试成绩更成线性关系,若男生平时成绩 不错,期末考试成绩也不错。而女生对于这点则不是非常明显,尤其是在平时成绩及格 的情况下,平时成绩好并不意味着期末考试成绩一定会好。

### 4. 学生成绩点图

首先,设置因子,R代码如下所示:

```
> source<-read.csv("xscj.csv")
> attach(source)
> factor(cut(平时成绩,5))->pjcj
```

然后,分区段绘制全部学生平时成绩的点图:

```
> dotchart(table(pjcj))
```

结果如图6-20所示。

### 5. 产品销量三维图

首先,读入产品销量数据,R代码如下:

```
> goods<-read.csv("goods.csv")
> goods
  地区编码 月份 销量
1         1   1 1200
2         1   2 3210
3         1   3  123
4         1   4 1111
```

```

5      1      5 688
6      1      6 2110
7      1      7 1123
8      1      8 6894
9      1      9 1470
10     1     10 1071
11     1     11 2250
12     1     12 1241
13     2      1 2222
14     2      2 1500
15     2      3 3200
16     2      4 1580
17     2      5 5562
.....
.....
>attach(goods)

```

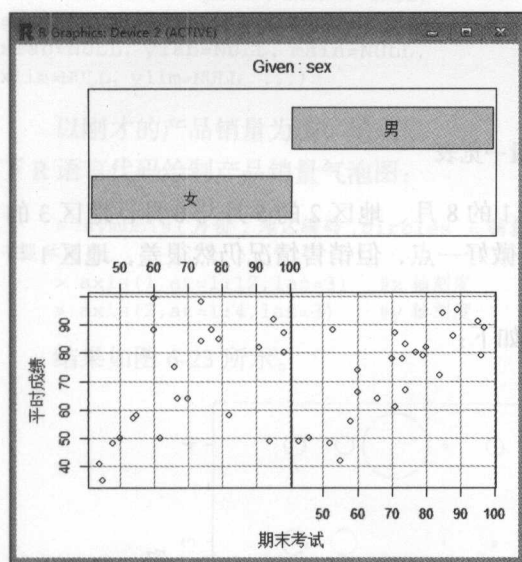


图 6-19 成绩协同图

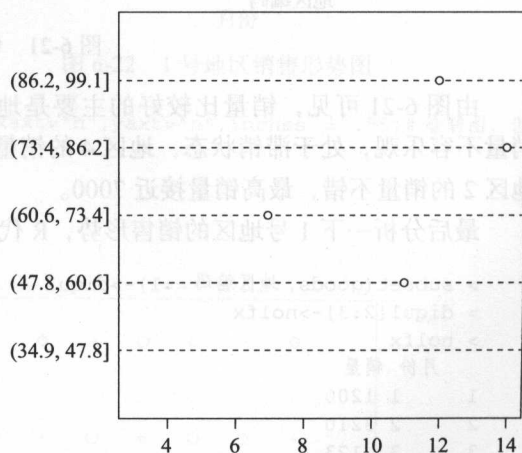


图 6-20 平时成绩的点图

然后, 检查是否安装 `scatterplot3d` 库, 安装过程如下:

```

> source("http://bioconductor.org/biocLite.R")
> biocLite("scatterplot3d")

```

再然后, 显示三维数据图:

```

> library(scatterplot3d)
> scatterplot3d(地区编码, 月份, 销量, highlight.3d=TRUE, pch=20, col.axis="blue",
col.grid="lightblue", grid=TRUE, type="h", main="销量一览表", lab=c(4,12))

```

结果如图 6-21 所示。

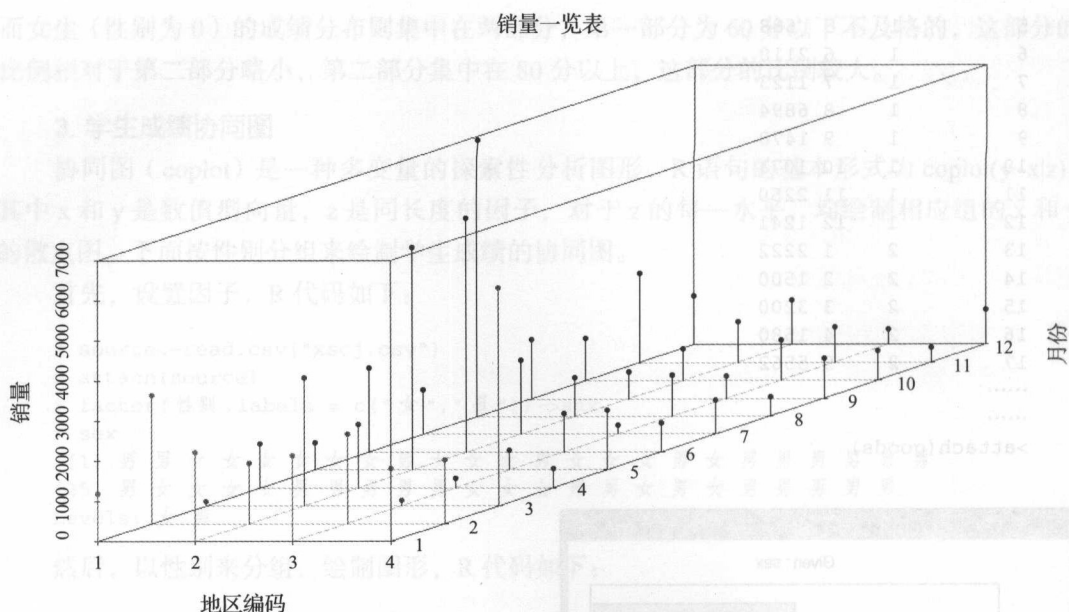


图 6-21 销量一览表

由图 6-21 可见，销量比较好的主要是地区 1 的 8 月、地区 2 的 5 月与 6 月，地区 3 的销量不容乐观，处于滞销状态，地区 4 的销量稍微好一点，但销售情况仍然很差，地区 1 与地区 2 的销量不错，最高销量接近 7000。

最后分析一下 1 号地区的销售形势，R 代码如下：

```
> subset(goods, 地区编码 == 1) -> diqu1
> diqu1[2:3] -> nolfx
> nolfx
  月份 销量
1    1 1200
2    2 3210
3    3  123
4    4 1111
5    5  688
6    6 2110
7    7 1123
8    8 6894
9    9 1470
10   10 1071
11   11 2250
12   12 1241
> plot(nolfx, type="o", main="1 号地区形势 ")
> abline(h=mean(nolfx$销量))
> axis(4, mean(nolfx$销量))
```

结果如图 6-22 所示。

观察图 6-22 可以看出，1 号地区在 2 月和 8 月的销售情况不错，而 3 月和 5 月的销售形

势则不容乐观。

## 6. 产品销量气泡图

气泡图是一个将点表示为圆圈的散点图，与 XY 散点图类似，但可表现的数据信息量更多，最典型的应用是通过更改气泡的大小和颜色，使数据的探索更加方便。在 R 语言中，用 `symbols()` 函数作气泡图，具体的调用格式如下：

```
Symbols(x, y=NULL, circles,
squares, rectangles, stars,
thermometers, boxplots, inches=TRUE,
add=FALSE, fg=par("col"), bg=NA,
xlab=NULL, ylab=NULL, main=NULL,
xlim=NULL, ylim=NULL, ...)
```

以刚才的产品销量为例，输入以下 R 语言代码绘制产品销量气泡图：

```
> symbols(月份, 地区编码, circles = 销量, xaxt="n", yaxt="n", inches = .55) # 绘制图，但不显示刻度。
```

```
> axis(1, at=1:12, las=3) # x 轴刻度
```

```
> axis(2, at=1:4, las=3) # y 轴刻度
```

结果如图 6-23 所示。

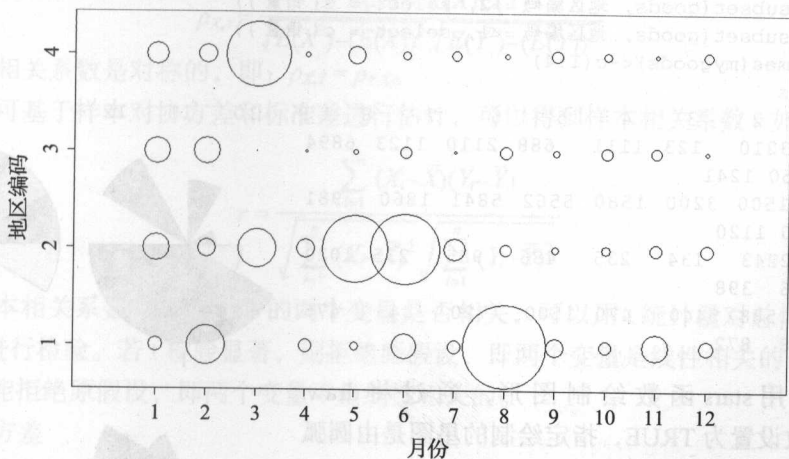


图 6-23 产品销量气泡图

观察图 6-23 可以看出，代表 1 号地区 8 月销量的气泡最大，表示销量最好，2 号地区的 5 月和 6 月的销量差不多，且都比较大。

## 7. 产品销量星图

星图的绘制方法如下:

1) 若设变量数目为  $n$  个, 则将圆周  $n$  等分, 连接圆心和这  $n$  个分点, 将形成  $n$  条半径, 这些半径依次定义为变量的坐标轴, 标以适当的刻度。

2) 对给定的一次观测值, 把  $n$  个变量值分别取在相应的坐标轴上, 将它们连接成  $n$  个圆弧。

下面使用星图来分析 4 个地区各月份的销量, R 代码如下:

首先, 调用和整理数据。

```
> goods<-read.csv("goods.csv")
```

```
> goods
```

	地区编码	月份	销量
1	1	1	1200
2	1	2	3210
3	1	3	123
4	1	4	1111
5	1	5	688
6	1	6	2110
7	1	7	1123
8	1	8	6894
9	1	9	1470
10	1	10	1071
11	1	11	2250
12	1	12	1241
13	2	1	2222
14	2	2	1500

```
> yue4<-subset(goods, 地区编码==4, select = c(销量))
```

```
> yue3<-subset(goods, 地区编码==3, select = c(销量))
```

```
> yue2<-subset(goods, 地区编码==2, select = c(销量))
```

```
> yue1<-subset(goods, 地区编码==1, select = c(销量))
```

```
> row.names(mygoods)<-c(1:4)
```

```
> mygoods
```

	1	2	3	4	5	6	7	8	9	10	11	12
1	1200	3210	123	1111	688	2110	1123	6894				
1470	1071	2250	1241									
2	2222	1500	3200	1580	5562	5841	1860	981				
658	789	1020	1120									
3	2144	2243	134	235	486	985	235	1020				
558	995	886	398									
4	1820	1588	5440	470	1500	720	845	476				
984	745	368	872									

然后调用 stars 函数绘制图形, 通过将 draw.segments 参数设置为 TRUE, 指定绘制的星图是由圆弧连接而成的。

```
> stars(mygoods, draw.segments=TRUE)
```

结果如图 6-24 所示。

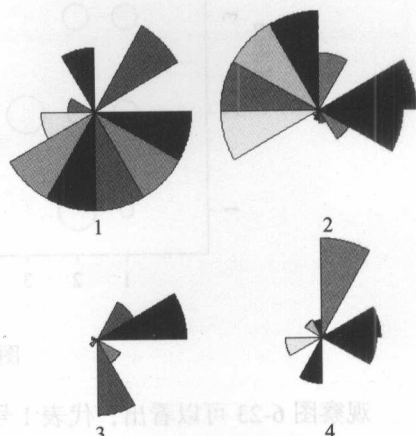


图 6-24 产品销量星图



观察图 6-24 可以看出, 1 号地区和 2 号地区的销量最好。

## 6.4.2 多元数据相关性分析

### 1. 皮尔森相关系数与协方差

#### (1) 皮尔森相关系数

皮尔森相关系数 (Pearson correlation coefficient) 也称皮尔森积矩相关系数, 是一种线性相关系数, 皮尔森相关系数是用来反映两个变量线性相关程度的统计量, 用于度量两个变量  $X$  和  $Y$  之间的相关 (线性相关), 如表 6-4 所示。其值介于  $-1 \sim 1$  之间, 负数为负相关, 正数为正相关。

表 6-4 皮尔森相关系数

皮尔森相关系数负值	皮尔森相关系数正值	相关性
$-0.09 \sim 0.0$	$0.0 \sim 0.09$	无
$-0.3 \sim -0.1$	$0.1 \sim 0.3$	弱
$-0.5 \sim -0.3$	$0.3 \sim 0.5$	中
$-1.0 \sim -0.5$	$0.5 \sim 1.0$	强

皮尔森相关系数的计算公式如下:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

上式中, 分子是协方差, 分子是两个变量标准差的乘积,  $X$  和  $Y$  的标准差都不为 0。

由于  $\mu_X = E(X)$ ,  $\sigma_X^2 = E[(X-E(X))^2] = E(X^2) - E^2(X)$ ,  $Y$  也类似, 并且

$$E[(X-E(X))(Y-E(Y))] = E(XY) - E(X)E(Y)$$

因此, 也可将皮尔森相关系数的计算公式写成如下形式:

$$\rho_{X,Y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - (E(X))^2} \sqrt{E(Y^2) - (E(Y))^2}}$$

皮尔森相关系数是对称的, 即:  $\rho_{X,Y} = \rho_{Y,X}$ 。

此外, 可基于样本对协方差和标准差进行估计, 可以得到样本相关系数  $r$  如下:

$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

利用样本相关系数推断总体中的两个变量是否相关, 可以用  $t$  统计量对总体相关系数为 0 的原假设进行检验。若  $t$  检验显著, 则拒绝原假设, 即两个变量是线性相关的; 若  $t$  检验不显著, 则不能拒绝原假设, 即两个变量不是线性相关的。

#### (2) 协方差

协方差表示的是两个变量总体误差的方差, 这与只表示一个变量误差的方差不同。如果两个变量的变化趋势一致, 也就是说如果其中一个大于自身的期望值, 另外一个也大于自身的期望值, 那么这两个变量之间的协方差就是正值。如果两个变量的变化趋势相反, 即其中一个大于自身的期望值, 另外一个却小于自身的期望值, 那么这两个变量之间的协方差就是负值。

## (3) 实例剖析

下面以某商品的销量及原料分析数据为例，分析原料对某商品销量的影响，R 代码如下：

```
> read.csv("ABCgoods.csv")->mygoods
```

```
> mygoods
```

A 原料 B 原料 C 原料 商品销量

```
1 0.85 0.13 0.02 4500
```

```
2 0.33 0.23 0.44 1800
```

```
3 0.64 0.24 0.12 3900
```

```
4 0.38 0.12 0.50 1000
```

```
5 0.10 0.20 0.70 740
```

```
6 0.28 0.17 0.55 990
```

```
7 0.15 0.80 0.05 910
```

```
8 0.18 0.70 0.12 930
```

```
> cov(mygoods)->myanalysis.cov# cov 为协方差矩阵
```

```
> cor(mygoods)->myanalysis.cor# cor 为相关系数矩阵
```

```
> myanalysis.cov
```

	A 原料	B 原料	C 原料	商品销量
A 原料	0.06716964	-0.03470179	-0.03246786	368.2161
B 原料	-0.03470179	0.07174107	-0.03703929	-147.3554
C 原料	-0.03246786	-0.03703929	0.06950714	-220.8607
商品销量	368.21607143	-147.35535714	-220.86071429	2235941.0714

```
> myanalysis.cor
```

	A 原料	B 原料	C 原料	商品销量
A 原料	1.0000000	-0.4998980	-0.4751737	0.9501366
B 原料	-0.4998980	1.0000000	-0.5245223	-0.3679187
C 原料	-0.4751737	-0.5245223	1.0000000	-0.5602393
商品销量	0.9501366	-0.3679187	-0.5602393	1.0000000

在 R 中可调用 `cor.test` 进行检测，它默认采用 `pearson` 检验 (`method` 参数)，置信区间水平默认为 0.95(`conf.level`)，`p` 值  $< 0.05$  则拒绝原假设，并认为两变量线性相关。代码如下所示：

```
> cor.test(~A 原料 + B 原料, data=mygoods)
```

Pearson's product-moment correlation

data: A 原料 and B 原料

$t = -1.4138$ ,  $df = 6$ ,  $p\text{-value} = 0.2071$

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

-0.8907805 0.3161398

sample estimates:

cor

-0.499898

```
> cor.test(~A 原料 + C 原料, data=mygoods)
```

Pearson's product-moment correlation

data: A 原料 and C 原料

$t = -1.3228$ ,  $df = 6$ ,  $p\text{-value} = 0.2341$

alternative hypothesis: true correlation is not equal to 0

95 percent confidence interval:

-0.8838848 0.3450297

```
sample estimates:
```

```
cor
```

```
-0.4751737
```

```
> cor.test(~A 原料 + 商品销量, data=mygoods)
```

```
Pearson's product-moment correlation
```

```
data: A 原料 and 商品销量
```

```
t = 7.4634, df = 6, p-value = 0.0002985
```

```
alternative hypothesis: true correlation is not equal to 0
```

```
95 percent confidence interval:
```

```
0.7427838 0.9911796
```

```
sample estimates:
```

```
cor
```

```
0.9501366
```

```
> cor.test(~C 原料 + 商品销量, data=mygoods)
```

```
Pearson's product-moment correlation
```

```
data: C 原料 and 商品销量
```

```
t = -1.6567, df = 6, p-value = 0.1487
```

```
alternative hypothesis: true correlation is not equal to 0
```

```
95 percent confidence interval:
```

```
-0.9068866 0.2386486
```

```
sample estimates:
```

```
cor
```

```
-0.5602393
```

```
> cor.test(~B 原料 + 商品销量, data=mygoods)
```

```
Pearson's product-moment correlation
```

```
data: B 原料 and 商品销量
```

```
t = -0.9692, df = 6, p-value = 0.3699
```

```
alternative hypothesis: true correlation is not equal to 0
```

```
95 percent confidence interval:
```

```
-0.8517618 0.4546201
```

```
sample estimates:
```

```
cor
```

```
-0.3679187
```

```
> cor.test(~B 原料 + C 原料, data=mygoods)
```

```
Pearson's product-moment correlation
```

```
data: B 原料 and C 原料
```

```
t = -1.5091, df = 6, p-value = 0.182
```

```
alternative hypothesis: true correlation is not equal to 0
```

```
95 percent confidence interval:
```

```
-0.8974739 0.2857795
```

```
sample estimates:
```

```
cor
```

-0.5245223

分析以上 cor.test 的调用结果，可以看出：

□ A 原料、B 原料、C 原料互相线性无关，应属于不需要按指定配比配置的。

□ A 原料与商品销量线性相关。

2. 影响因素分组

下面利用相关性分析，来分析某类商品的网络销售情况，并将影响因素分组。

首先，读入数据：

```
> read.csv("sales2.csv")->mysales
> mysales
  价格 好评率 月平均评论数 包装精美程度 .1.3. 品牌知名度 .1.3. 月平均销量
1    50     94         150          3          3          350
2   150     68         120          2          2          290
3   190     88         100          2          2          160
4  1500     85           5          1          1           40
5    69     98          40          2          2           64
6   800     73           3          2          1           20
7    32     88         180          3          3          400
8   500     90           6          2          1           10
9   182     68          19          2          2           70
10   23     89         190          3          2          500
```

然后，计算相关系数：

```
> cor(mysales)
           价格      好评率 月平均评论数 包装精美程度 .1.3.
价格      1.0000000 -0.1648994 -0.6538897 -0.7769090
好评率    -0.1648994  1.0000000  0.2529134  0.2933271
月平均评论数 -0.6538897 0.2529134  1.0000000  0.8197505
包装精美程度 .1.3. -0.7769090 0.2933271  0.8197505  1.0000000
品牌知名度 .1.3. -0.7531132 0.2556607  0.7921685  0.7619048
月平均销量   -0.5897317 0.1777546  0.9798662  0.8104570
           品牌知名度 .1.3. 月平均销量
价格      -0.7531132 -0.5897317
好评率      0.2556607  0.1777546
月平均评论数 0.7921685  0.9798662
包装精美程度 .1.3.  0.7619048  0.8104570
品牌知名度 .1.3.  1.0000000  0.7291934
月平均销量    0.7291934  1.0000000
```

再然后，对各个指标的相关度进行分析。按相关度将指标进行分组，使相关系数高的指标归为同一组，找到出现除开 1 以外的绝对值最大相关度——月平均评论数与月平均销量的相关系数为 0.979 866 2，包装精美程度 .1.3. 与月平均销量的相关系数为 0.810 457 0，将这 3 个指标归为一组，然后找到价格与品牌知名度的相关系数 0.753 113 2，将这两个指标归为一组，好评率为一组，这样一共分为了三组。

将每一个组内部的每个指标的权值设为不同的值（保证权值之和为 1），分别设置如下：

1) 第 1 组中月平均销量的权值为 0.8，月平均评论数的权值为 0.15，包装精美程



度 .1.3. 的权值为 0.05 (计算时为 3- 包装精美程度, 因为包装精美程度数字越小, 表示越精美, 指标是这样设计的)。

2) 第 2 组, 价格的权值为 0.8, 品牌知名度 .1.3. 的权值为 0.2 (计算时为 3- 品牌知名度, 因为品牌知名度数字越小, 表示越知名, 指标是这样设计的)

3) 第 3 组就一个指标好评率。不设权值。

最后, 计算这 3 组的得分。

```
> attach(mysales)
> group1<-( 月平均评论数 *0.15+(3- 包装精美程度 .1.3.) *0.05+ 月平均销量 *0.8)/3
> group2<-( (3- 品牌知名度 .1.3.) *0.2+ 价格 *0.8)/2
> group3<- 好评率
> myavg=(group1*0.45+group2*0.1+group3*0.45)/3
> names(myavg)<-c(1:10)
> sort(myavg,decreasing=TRUE)
```

10	4	7	1	2	3	6	8
35.08500	34.39917	30.97667	29.89167	24.70583	22.88917	22.44833	20.62083
5	9						
18.48583	15.57500						

分析 sort 函数的结果, 可看出, 按综合排名, 从高到低的商品序号如下:

10	4	7	1	2	3	6	8	5	9
----	---	---	---	---	---	---	---	---	---

## 6.5 小结

对于统计中的描述性分析来说, 最好的学习方法就是实践, 光看理论比较难于理解。本章首先以实例的形式分析了正态分布函数、峰度系数、累计分布概率、概率密度函数及曲线、分位点、正态检验与分布拟合等数据统计指标; 然后介绍了正态分布等分布模型; 最后, 讲解了多变量数据分析和可视化、数据相关性分析等知识。

## 思考题

(1) 下载本书例子中的美国地震台数据 earthquakes.csv, 对于其中震深 Depth 进行分析, 绘制累计分布概率的散点图。

(2) 下载本书例子中的美国地震台数据 earthquakes.csv, 对于其中震级 Magnitude 进行分析, 分析概率密度。

(3) 下载本书例子中的全国就业调查情况数据 youxiangz.csv, 分析“平均劳动报酬”的正态分析函数、峰度系数、累计分布概率、概率密度函数及曲线、直方图。

(4) 下载本书例子中的商品销售情况数据 sales2.csv, 以价格为 X 轴, 以月平均评论数为 Y 轴, 绘制关于月平均销量的气泡图, 分析价格和月平均评论数处于哪些区间内时, 月平均销量较好。

(5) 下载本书例子中的学生成绩数据 xscj.csv, 以性别、学号、期末考试为指标, 绘制关于期末考试的三维图, 分析哪些学生成绩比较好。



## 假设检验与回归模型案例

### 7.1 假设检验

假设检验是除参数估计之外的另一类重要的统计推断问题，它认为小概率事件在一次试验中几乎是不可能发生的，即：对总体的某个假设是真实的，那么不利于或不能支持这一假设的事件在一次试验中几乎是不可能发生的；要是在一次试验中小概率事件发生了，那么就有理由怀疑这一假设的真实性，拒绝这一假设。具体来说，想检验其正确性的称为零假设（null hypothesis），零假设通常反映的是研究者对未知参数的看法，相对于零假设的其他论述则是对立假设（alternative hypothesis），它反映了执行检验者对参数可能数值的对立的看法，也就是说，对立假设通常才是研究者最想知道的。

假设检验的过程，可以用法庭审理的例子来说明：如果现在法庭上有一名被告，假设该被告是清白的，那么检察官必须要提出足够的证据证明被告的确有罪。在证明被告有罪前，被告是被假设为清白的，假设被告是清白的这个假设，就相当于零假设，假设被告是有罪的这个假设，则是对立假设。检察官提出的证据，是否足以确定该被告有罪，则需要经过检验。

#### 7.1.1 二项分布假设检验

考察由  $n$  次随机试验组成的随机现象，它要同时满足以下条件：

- 重复进行  $n$  次随机试验。
- $n$  次试验相互独立。
- 每次试验仅有两个可能的结果。
- 每次试验成功的概率为  $p$ ，失败的概率为  $1-p$ 。

在上述四个条件下，假设  $X$  表示  $n$  次独立重复试验中成功出现的次数，显然  $X$  是可以取

0, 1, ...,  $n$  等  $n+1$  个值的离散随机变量, 这个分布称为二项分布, 记为  $B(n, p)$ 。

在二项分布中, 设有  $k$  次成功和  $n-k$  次失败,  $k$  次成功可以出现于  $n$  次试验的任何一次,  $n$  次试验中正好得到  $k$  次成功的概率如下:

$$\Pr(K=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

在假设检验中, 经常遇到非正态总体的统计数据, 这类数据可使用二项分布的总体假设检验方法, 下面以实例进行讲解。

### 1. 游戏策略调整

某游戏的某区域内经常发生暴力 PK 事件, 从而给在这个区域内做任务的新手玩家和某些老玩家带来了很多困扰, 以往发生这类事件的概率为 25%, 对这个区域的游戏策略进行调整后, 随机抽取了 300 多个玩家进行测试, 结果有 20 个被迫卷入暴力 PK, 那么这个游戏策略的调整是否有效?

```
> binom.test(x=20,n=300,p=0.25,alternative="less")
```

Exact binomial test

data: 20 and 300

number of successes = 20, number of trials = 300, p-value < 2.2e-16

alternative hypothesis: true probability of success is less than 0.25

95 percent confidence interval:

0.00000000 0.09540198

sample estimates:

probability of success

0.06666667

这里将原假设  $H_0$  设为  $p \geq 0.25$ , 而备择假设  $H_1$  设为  $p < 0.25$ 。观察以上结果, 在 95% 的置信率基础上,  $p$  值  $< 2.2e-16 < 0.05$ , 因此可以认为该游戏策略的调整对维护这个区域的秩序起到了一定的效果。

### 2. 游戏宝石出产检测

某网络游戏中有一区域为宝石采矿区, 玩家在此开采宝石的比例应为高级宝石: 中级宝石: 低级宝石 = 3 : 11 : 23, 抽取 671 个玩家的采矿记录, 开采数量为 70 : 190 : 411, 请问实际出产比例是否符合理论要求?

```
> chisq.test(c(70,190,411),p=c(3,11,23)/37)
```

Chi-squared test for given probabilities

data: c(70, 190, 411)

X-squared = 5.0106, df = 2, p-value = 0.08165

观察以上结果,  $p$  值为 0.08165, 结果大于 0.05 接受原假设, 实际出产比例符合理论要求。

## 7.1.2 数据分布检验

### 1. 正态分布检测

使用 `shapiro.test` 函数可检测数据的正态分布。例如，设显著性水平为 0.05，检查以下数据是否为正态分布。

```
12, 22, 67, 89, 56, 10, 124, 235, 77, 88, 66, 79, 80, 82
```

R 代码如下：

```
> x<-c(12,22,67,89,56,10,124,235,77,88,66,79,80,82)
> shapiro.test(x)
```

Shapiro-Wilk normality test

data: x

W = 0.8173, p-value = 0.008236

观察以上结果， $p$  值小于显著性水平 0.05，因此可拒绝原假设，因为原假设为符合正态分布，因此可认为该数据不符合正态分布。

再来看一个例子，如下面 R 代码所示：

```
> shapiro.test(rnorm(100, mean = 5, sd = 3))
```

Shapiro-Wilk normality test

data: rnorm(100, mean = 5, sd = 3)

W = 0.9914, p-value = 0.7787

观察以上结果， $p$  值大于显著性水平 0.05，不能拒绝原假设，因此可认为该数据符合正态分布。

### 2. Kolmogorov-Smirnov 检验

Kolmogorov-Smirnov 检验 (K-S 检验) 基于累积分布函数，用于检验一个经验分布是否符合某种理论分布或比较两个经验分布是否有显著性差异。下面以对某后台服务程序的稳定性检测为例进行讲解。

现对某服务器的某后台服务程序进行稳定性检测，记录 8 次无故障稳定工作的小时数，分别为：240、180、320、190、160、60、400、340，经估计它符合  $\lambda=1/290$  的指数分布，下面来验证一下稳定性的分布：

```
> ks.test(servtime, "pexp", 1/290)
```

One-sample Kolmogorov-Smirnov test

data: servtime

D = 0.299, p-value = 0.394

alternative hypothesis: two-sided

观察以上结果， $p$  值  $>0.05$ ，无法拒绝原假设，因此无故障稳定工作小时数符合该分布。  
提示：R 语言中，`pexp` 是一个指数分布函数，指数分布有以下系列的 R 函数：

- `dexp` 给出了密度：`dpexp(x,rate=1,t=0,log=FALSE)`
- `pexp` 给出了分布函数：`ppexp(q,rate=1,t=0,lower.tail=TRUE,log.p=FALSE)`
- `qexp` 给出了分位数功能：`qpexp(p,rate=1,t=0,lower.tail=TRUE,log.p=FALSE)`
- `rexp` 给出了随机产生的偏离：`rpexp(n,rate=1,t=0)`

7.1.3 正态总体均值检验

假设某游戏服务器接受游戏客户端发来的报告，内容是某场景载入时间的报告（载入时间符合正态分布），平均载入时间要求小于 225。现提取报告的部分数据，检验载入时间是否正常，设显著性水平为 0.05，用 R 语言分析，代码如下（ $x$  为时间）：

```
> x<-c(220,218,210,220,215,221,212,225,209,230,180,182,150,190,230,227,240,225)
> t.test(x, alternative = "less", mu = 225)
```

One Sample t-test		显著性	置信
data: x		0.05	0.001>
t = -2.5922, df = 17, p-value = 0.009493		0.01	0.00E - 0.001
alternative hypothesis: true mean is less than 225		0.01	0.00E - 0.001
95 percent confidence interval:		0.01	0.00E - 0.001
-Inf 220.5051		0.01	0.00E - 0.001
sample estimates:			
mean of x			
211.3333			

观察上述结果， $p$  值为 0.009 493，小于显著性水平 0.05，有理由拒绝原假设，原假设为平均载入时间大于 225，因此，可选择备择假设，即：平均载入时间小于 225。

也可将平均载入时间大于 225 设为备择假设，而平均载入时间小于 225 设为原假设，用 R 语言分析，代码如下：

```
> t.test(x, alternative = "greater", mu = 225)
```

One Sample t-test		显著性	置信
data: x		0.05	0.001>
t = -2.5922, df = 17, p-value = 0.9905		0.01	0.00E - 0.001
alternative hypothesis: true mean is greater than 225		0.01	0.00E - 0.001
95 percent confidence interval:		0.01	0.00E - 0.001
202.1616 Inf		0.01	0.00E - 0.001
sample estimates:			
mean of x			
211.3333			

观察上述结果， $p$  值为 0.9905，大于显著性水平 0.05，无法拒绝原假设，原假设为平均载入时间小于 225，因此，可认为平均载入时间小于 225，载入时间正常。



7.1.4 列联表

列联表是观测数据按两个或更多个属性（定性变量）分类时所列出的频数表。比如：对随机抽取的 1000 人按性别（男或女）及色觉（正常或色盲）两个属性分类，可以得到如表 7-1 所示的二行二列的列联表。

表 7-1 性别与色觉列联表

色觉 \ 性别	男	女	行和
正常	442	514	956
色盲	38	6	44
列和	480	520	1000

下面以满意度列联表为例进行讲解，表 7-2 是某类商品价格与客户满意度的列联表。

表 7-2 满意度与价格列联表

满意度 \ 价格	不满意	比较满意	很满意	合计
<1000	20	56	34	110
1000 ~ 3000	10	33	21	64
>3000	32	66	24	122
合计	62	155	79	296

可通过列联表数据的独立性对价格与客户满意度之间的关系进行分析，在 R 语言中可通过 `chisq.test` 函数来完成检测：

```
> c(20,56,34,10,33,21,32,66,24)->x
> c(3,3)->dim(x)
> x
      [,1] [,2] [,3]
[1,]   20   10   32
[2,]   56   33   66
[3,]   34   21   24
>
> chisq.test(x)
Pearson's Chi-squared test
data:  x
X-squared = 6.8985, df = 4, p-value = 0.1413
```

观察以上结果， $p>0.05$  说明不拒绝原假设，两个变量独立无关，价格与客户的满意度之间没有关系。

此外，当列联表中有频数低于 4 时，最好使用 `fisher` 进行精确检测。例如：某网上商场对某种高档商品进行了一次短期的测试，验证自动导购系统的有效性，商场分别提供了普通购买通道和自动导购系统通道，表 7-3 为该测试的列联表。



表 7-3 高档商品测试列联表

是否购买 通道	成功购买	未购买	合计
自动导购系统通道	11	9	20
普通购买通道	3	12	15
合计	14	26	35

对表 7-3 的数据进行 fisher 检测，R 语言代码如下：

```
> c(11,3,9,12)->x
> dim(x)<-c(2,2)
> fisher.test(x)
```

Fisher's Exact Test for Count Data

```
data: x
p-value = 0.04614
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.8730177 33.8560814
sample estimates:
odds ratio
 4.662271
```

>

观察以上结果， $p < 0.05$  拒绝原假设，因此认为自动导购系统有效，odds ratio > 1 表示存在正相关关系，即通过自动导购系统通道进入的顾客越多，该商品的购买率就越大。

### 7.1.5 符号检测

符号检测通过观察样本值与总体中某个位置（比如中位数）的差实现，将样本观察值与总体中位数的差之间的关系用符号表示为大于、小于或正、负关系，从而进行检测。

#### 1. 某商场 VIP 客户信息推送

以某商场 VIP 客户广告推送为例，某商场需要针对某类商品建立该类商品的 VIP 大客户，以便定期向该类客户推送相关广告。客户服务部门推荐了客户 A，从数据库中随机抽取了 100 个客户资料，统计他们前 4 个季度的平均季消费数据（在这里用平均随机数模拟数据），检测客户 A 的平均季消费是否处于中上水平（位于中位数以上）。首先，构造数据，R 语言代码如下：

```
> sample(200:50000,100)->sale
> sale
[1] 8447 13987 8809 44437 22973 28093 30594 28060 21101 45155 36128 30129
[13] 556 33977 9283 35094 903 32885 11639 15533 29150 47368 4993 5376
[25] 1869 15975 25120 33530 31767 41845 39623 3586 22671 16128 14814 24993
[37] 45830 10349 43989 35650 45179 35282 27204 5485 22990 21475 14533 42852
[49] 15986 28411 16683 15832 27207 19062 10256 34549 46159 16315 43097 40038
[61] 27758 14936 26161 18694 25139 13208 26837 30171 13663 14082 46909 26498
```

```
[73] 7830 35810 15183 41769 8880 47928 13387 33231 28978 39486 6309 19344
[85] 12935 41976 13429 16291 31159 33646 1742 48160 43169 40165 38915 24941
[97] 25181 30077 19475 26836
```

```
>
```

然后,使用 `sum(sale>29900)` 表示所有大于该客户平均消费水平的样本数,进行 `binom.test` 检测:

```
> binom.test(sum(sale>29900),length(sale),al="less")
```

```
Exact binomial test
```

```
data: sum(sale > 29900) and length(sale)
```

```
number of successes = 38, number of trials = 100, p-value = 0.01049
```

```
alternative hypothesis: true probability of success is less than 0.5
```

```
95 percent confidence interval:
```

```
0.0000000 0.4667535
```

```
sample estimates:
```

```
probability of success
```

```
0.38
```

```
>
```

```
95 percent confidence interval:
```

```
0.0000000 0.4667535
```

这里, `binom.test` 使用的是二项分布  $B(100, 1/2)$  的检测, 每次抽样要么  $M \geq M_0$ , 要么  $M < M_0$ , 成功的概率为  $1/2$ , 即在中位数以上(含中位数)和中位数以下的概率均为  $0.5$ 。单侧区间估计的上界为  $0.4667535$ , 小于原假设出现的概率  $0.5$ , 因此拒绝了原假设。

本例中的假设为:  $H_0: M \geq M_0, H_1: M < M_0$ 。其中,  $M_0$  为该客户的消费金额,  $M$  为样本的中位数, 代表平均消费水平, 原假设为平均消费水平比该客户的消费金额大, 备择假设为平均消费水平比该客户的消费金额小, 如果备择假设成立, 则代表拒绝原假设, 且该客户在这类商品的消费能力处于中上。

最后, 观察 `binom.test` 分析结果的  $p$  值, 它等于  $0.01049$ , 小于  $0.05$ , 拒绝原假设, 再观察  $95\%$  的置信区间 ( $95\text{ percent confidence interval}$ ), 其上限为  $0.4667535$ , 低于  $0.5$ , 仍拒绝原假设, 因此, 该客户的消费金额高于该类商品的客户中间水平, 处于中上层消费, 可以考虑将其设为 VIP 客户。



**提示** `sum` 函数的参数可设置条件, 完成条件计数功能。比如下面的代码:

```
> x<-c(1,3,9,20,41)
```

```
> x>7
```

```
[1] FALSE FALSE TRUE TRUE TRUE
```

```
> sum(x>7)
```

```
[1] 3
```

## 2. 虚拟道具促销

某游戏公司对 X 游戏的某类虚拟道具进行了为期 6 周的促销测试, 以增加其销量, 采用

了以下两种促销手段, 请问这两种促销手段的效果相同吗, 如果效果相同, 则可以定期采用这两种促销手段, 搞类似的促销活动, 数据如表 7-4 所示。

表 7-4 相对未促销前两种促销手段所增加的销量

促销手段	第 1 周	第 2 周	第 3 周	第 4 周	第 5 周	第 6 周
A	50	60	45	62	48	59
B	39	68	58	64	52	61

编写 R 代码如下:


```
> a<-c(50,60,45,62,48,59)
> b<-c(39,68,58,64,52,61)
> binom.test(sum(a>b),length(a))
```

Exact binomial test

```
data: sum(a > b) and length(a)
number of successes = 1, number of trials = 6, p-value = 0.2188
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.004210745 0.641234579
sample estimates:
probability of success
 0.1666667
```

>

观察以上结果,  $p$  值为 0.166 666 7, 大于 0.05, 不能拒绝原假设, 原假设是促销手段 A 的销量增加数比促销手段 B 多, 且有明显差距, 因此可认为, A 比 B 的促销效果好。

 **提示**  $\text{sum}(a>b)$  表示  $a$  大于  $b$  的数量,  
 $\text{sum}(a<b)$  表示  $a$  小于  $b$  的数量。

### 3. 某商场顾客群分析

某网上商场对购买 A 类商品的顾客群进行分析, 发现他们更喜欢同时购买 B 类或 C 类商品, 下面随机抽取了其中的 20 个顾客作为样本, 购买 B 类商品的为  $b$ , 购买 C 类商品的为  $c$ , 1 表示购买, 0 表示没购买。部分数据如表 7-5 所示。(完整数据可参见本书源码包的 sale3.csv 文件。)

表 7-5 顾客群购买情况

$b$	$c$
1	0
1	0
0	1
1	0

用采牌效以同，同果果效同，同同果效同同手能分将网友同者，同手能分 (续)

b	c
1	0
1	0
1	0

可将原假设定为这类顾客群更喜欢同时购买 C 商品，而备择假设是更喜欢同时购买 B 商品。此外，以 95% 的置信度进行检测，显著水平取 0.05。编写如下 R 代码：


```
> read.csv("sale3.csv")->mysale
> c_count<-sum(mysale$b==0 & mysale$c==1)
> b_count<-sum(mysale$b==1 & mysale$c==0)
> binom.test(c_count,b_count+c_count,p=1/2,al="less",conf.level=0.95)

Exact binomial test

data:  c_count and b_count + c_count
number of successes = 4, number of trials = 17, p-value = 0.02452
alternative hypothesis: true probability of success is less than 0.5
95 percent confidence interval:
 0.0000000 0.4605494
sample estimates:
probability of success
 0.2352941

>
```

分析上述结果， $p = 0.024\ 52$ ， $p$  值小于  $0.05(100\%-95\%=5\%=0.05)$ ，可拒绝原假设（更喜欢同时购买 C 商品），而备择假设成立，购买 A 商品的客户更喜欢同时购买 B 商品。同时可以看到单侧区间上界为  $0.460\ 549\ 4$ （低于  $0.5$ ），这里也表明要拒绝原假设。

 **提示** 该案例中，同时购买 B 和 C 商品的客户不用列入样本容量进行计算。

### 7.1.6 秩相关检验

#### 1. spearman 秩相关检验

spearman 秩相关检验根据计算得到的秩统计量产生秩相关系数  $r$ ，相互独立时， $E(r)=0$ ；正相关时， $r$  为正值；负相关时， $r$  为负值；可检测分布无关性。下面对某商品每周的点击次数与每周的销量进行统计，表 7-6 是连续 6 周的统计数据。

表 7-6 每周的点击次数与销量

点击次数	310	480	190	400	590	520
销量	150	210	70	190	230	210



编写 R 代码如下:

```
> sales<-c(150,210,90,190,230,211)
> hits<-c(310,480,190,400,590,520)
> cor.test(hits,sales,method="spearman")

Spearman's rank correlation rho

data: hits and sales
S = 0, p-value = 0.002778
alternative hypothesis: true rho is not equal to 0
sample estimates:
rho
1
>
```

观察以上结果,  $p$  值  $0.002\ 778 < 0.05$ , 因此拒绝原假设, 认为两个变量是相关的。此外,  $\rho > 1$  表示正相关,  $\rho < 1$  表示负相关,  $E(\rho) = 0$  表示相互独立无关。综上所述, 每周的点击次数与每周的销量是正相关的, 点击的次数越多, 销量就越多。

## 2. Wilcoxon 秩检验

Wilcoxon 秩检验考虑了符号检测 (如样本观测值与总体中位数差的符号), 而且考虑了观测值与原假设某位置的具体差距的大小。

以购物网站的网页改版为例进行讲解, 假设购物网站对某类商品的访问网页进行了改版, 并能同时保留改版前的网页和改版后的网页供顾客选择浏览, 通过一段时间的测试后, 对通过两种网页进行购买的行为进行分析, 抽取其中的 7 个商品进行检测, 查看改版的效果。销量如表 7-7 所示。

表 7-7 网页改版前后的销量数据

某类商品的所有编号	网页改版前的销量	网页改版后的销量
10001	380	530
10002	410	500
10003	290	506
10004	375	497
10005	186	280
10006	300	388
10007	60	124

首先, 检测网页改版后该类商品的销量提升是否达到了期待的水平, 即销量中位数在 400 以上。设定原假设为销量提升达到期待水平, 该类商品中有一半商品的销量在 400 以上 (即销量中位数在 400 以上), 而备择假设为销量中位数在 400 以下。R 代码如下:

```
> sales<-c(530,500,506,497,280,388,124)
```



```
> wilcox.test(sales, mu=400, alternative="less", correct=FALSE, exact=FALSE, conf.
int=TRUE)
```

```
Wilcoxon signed rank test
```

```
data: sales
V = 15, p-value = 0.5671
alternative hypothesis: true location is less than 400
95 percent confidence interval:
 -Inf 506.0001
sample estimates:
(pseudo)median
 406.0668
```

```
>
```

$p$  值为 0.5671,  $P > 0.05$ , 不能拒绝原假设, 再观察置信度为 95% 的区间的估计上限为 506.001。因此, 销量中位数在改版后达到了 400 以上。



提示 以上代码中, **alternative** 表示备择假设, **correct** 表示在计算  $P$  值时是否采用连续性修正, **exact** 表示是否精确计算  $P$  值, **conf.int** 设定是否输出置信区间。

那能不能再乐观些, 销量中位数能否达到 600 以上呢? 编写如下 R 代码进行检测:

```
> wilcox.test(sales, mu=600, alternative="less", correct=FALSE, exact=FALSE, conf.
int=TRUE)
```

```
Wilcoxon signed rank test
```

```
data: sales
V = 0, p-value = 0.00898
alternative hypothesis: true location is less than 600
95 percent confidence interval:
 -Inf 506.0001
sample estimates:
(pseudo)median
 406.0668
```

```
>
```

$p=0.00898$  小于 0.05, 显然需要拒绝原假设 (销量中位数大于 600), 不能如此乐观地将销量中位数估计到 600 以上。


然后, 验证新版网页是否提高了该类商品的销量。R 代码如下:

```
> salesnew<-c(530,500,506,497,280,388,124)
> salesold<-c(380,410,290,375,186,300,60)
> wilcox.test(salesnew,salesold,alternative="greater",paired=TRUE)
```

```
Wilcoxon signed rank test
```

```
data: salesnew and salesold
V = 28, p-value = 0.007813
alternative hypothesis: true location shift is greater than 0
```

观察上述结果,可以看出,原假设为新版网页与老版网页效果相同,备择假设为新版网页能提高销量。 $p=0.007813$ ,小于0.05,因此拒绝原假设,新版网页能提高该类商品的销量。

 **提示** paired 参数表示样本是否属于成对样本, TRUE 表示是成对样本。

下面以某游戏的区域策略调整为例,假设某游戏在某区域内调整了策略(更新地图、怪物等),使该区域相对于B职业玩家来说,更适合于A职业玩家进行升级和活动,游戏开发商对在该游戏区域内活动的A职业玩家和B职业玩家采取了随机有奖问卷方式,收集该区域的活动满意度,调查评分(满分为100分)如表7-8所示。

表 7-8 两类职业玩家满意度调查表

A 职业玩家	80	96	84	79	81	92	75	85
B 职业玩家	68	75	71	92	66			

将原假设  $H_0$  设为调整策略后,A职业玩家与B职业玩家无差异,备择假设  $H_1$  是A职业玩家比B职业玩家更满意。R代码如下:

```
> a<-c(80,96,84,79,81,92,75,85)
> b<-c(68,75,71,92,66)
> wilcox.test(a,b,alternative="greater",exact=FALSE,correct=TRUE)
```

Wilcoxon rank sum test with continuity correction

```
data: a and b
W = 33, p-value = 0.03326
alternative hypothesis: true location shift is greater than 0
```

观察上述检测结果, $p=0.03326<0.05$ ,因此拒绝原假设,游戏策略调整效果较显著,A职业玩家更愿意在该区域内活动和升级。

### 7.1.7 Kendall 相关检验

Kendall 相关检验通过协同进行检测,设  $x$  和  $y$  两个变量进行检测,如果  $(x_j-x_i)(y_j-y_i)>0$ ,则称对子  $(x_i, y_i)$ 、 $(x_j, y_j)$  有同样的倾向;反之,如果  $(x_j-x_i)(y_j-y_i)<0$ ,则称对子是不协同的。通过将协同对子代入 Kendall 相关系数,再计算变量之间的相关性。对表 7-6 的数据进行分析,首先,构造数据。R代码如下:

```
> sales<-c(150,210,90,190,230,211)
```

```
> hits<-c(310,480,190,400,590,520)
```

然后,将 method 参数设为 kendall, 进行分析:

```
> cor.test(hits,sales,method="kendall")
```

```
Kendall's rank correlation tau
```

```
data: hits and sales
```

```
T = 15, p-value = 0.002778
```

```
alternative hypothesis: true tau is not equal to 0
```

```
sample estimates:
```

```
tau
```

```
1
```

观察以上结果,结论仍是正相关, $p$  值 0.002 778 小于 0.05,因此拒绝原假设。

## 7.2 回归模型

回归是研究一个随机变量  $Y$  对另一个变量 ( $X$ ) 或一组变量 ( $X_1, X_2, \dots, X_k$ ) 的相依关系的统计分析方法,它通过规定因变量和自变量的关系来确定变量之间的因果关系,建立回归模型,并根据实测数据来求解模型的各个参数,并评价回归模型是否能够很好地拟合实测数据。

### 7.2.1 回归预测与显著性检验

下面按方程  $y = \beta_0 + \beta_1 x + \varepsilon$  进行一元线性回归和预测。

首先,建立回归模型。R 代码如下:

```
> x<-c(12,9,3,7,17,19)
```

```
> y<-c(28,20,8,13,36,39)
```

```
> lm(y~1+x)->mylm.lm
```

```
> mylm.lm
```

```
Call:
```

```
lm(formula = y ~ 1 + x)
```

```
Coefficients:
```

```
(Intercept)      x  
1.284        2.034
```

```
> summary(mylm.lm)
```

```
Call:
```

```
lm(formula = y ~ 1 + x)
```

```
Residuals:
```

```
1      2      3      4      5      6  
2.3048 0.4076 0.6132 -2.5239 0.1335 -0.9351
```

```
Coefficients:
```

```

      Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.2840      1.6609   0.773 0.482617
x            2.0343      0.1332  15.273 0.000107 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

Residual standard error: 1.811 on 4 degrees of freedom
Multiple R-squared:  0.9831,    Adjusted R-squared:  0.9789
F-statistic: 233.3 on 1 and 4 DF,  p-value: 0.0001072

```

观察以上结果,从 summary 给出的回归模型信息可以看出,Residuals 部分指出  $x_6$  个值的残差,Coefficients 部分指出  $\beta_0$  预测为 1.2840,其标准差(Std. Error)为 1.6609; $x$  的参数( $\beta_1$  部分)预测为 2.0343,其标准差(Std. Error)为 0.1332。

然后,指定自变量(如  $x$  值)后,对因变量(如  $y$  值)进行预测,R 代码如下:

```

> myx<-data.frame(x=c(20,28,19))
> predict(mylm.lm,myx,interval="prediction",level=0.95)
      fit      lwr      upr
1 41.96934 35.63207 48.30661
2 58.24346 49.98260 66.50432
3 39.93508 33.78026 46.08989
>

```

观察以上结果,在置信度为 95% 的情况下,如果  $x=20$ ,则  $y$  值预测为 41.969 34, $y$  值预测落在区间 [35.632 07,48.306 61],如果  $x=19$ ,则  $y$  值预测落在区间 [33.780 26, 46.089 89]。

此外,还可对回归模型中的参数进行预测。

以一元线性回归模型  $y=\beta_0+\beta_1x+\varepsilon$  为例, $\varepsilon$  是随机误差, $\beta_0$  和  $\beta_1$  是参数。设  $\alpha$  为显著水平,则置信度为  $1-\alpha$ ,其参数的区间估计如下:

$$[\beta - sd(\beta) t_{\frac{\alpha}{2}}(n-2), \beta + sd(\beta) t_{\frac{\alpha}{2}}(n-2)]$$

按照上述公式,计算上述回归模型参数在 95% 置信度(显著水平  $\alpha=0.05$ ) 的预计区间,R 代码如下:

```

> summary(mylm.lm)->mylm.summary
># 取出参数预测部分
> mylm.summary$coefficients->mycof
> mycof
      Estimate Std. Error    t value    Pr(>|t|)
(Intercept)  1.284040   1.6609290   0.7730852 0.4826172233
x            2.034265   0.1331944  15.2729058 0.0001071897
# 显著水平 a
> a<-0.05
># mylm.lm$df.residual 为模型的自由度,其值为 n-2, n 为 x 向量或 y 向量包括元素的个数.
> mylm.lm$df.residual->mydf
> lower_bound<-mycof[,1]-mycof[,2]*qt(1-a/2,mydf)
> upper_bound<-mycof[,1]+mycof[,2]*qt(1-a/2,mydf)
> dimnames(mycof)[[1]]->myrowname
> c(" 预计值 "," 下界值 "," 上界值 ")->mycolname
># 显示参数预测区间

```

```
> matrix(c(mycof[,1],lower_bound,upper_bound),ncol=3,dimnames=list(myrowname,my
colname))
          估计值    下界值    上界值
(Intercept) 1.284040 -3.327439 5.895518
x           2.034265  1.664458 2.404072
>
```

分析以上结果,  $x$  的系数参数预计在 [1.664 458, 2.404 072] 的范围内。

## 7.2.2 回归诊断

通过对回归进行显著性检验, 可检测回归效果, 尤其是在多元线性回归分析中, 无法用散点图直观地判断众多变量  $x_1, x_2, \dots, x_{j-1}, x_{j+1}, \dots, x_p$  与  $y$  之间是否存在线性关系的时候。回归效果的显著性检验可分为回归方程的显著性检验和回归系数的显著性检验。

以 `lm` 函数结果为参数, 调用 `summary` 函数, 将返回很多回归检验分析信息。我们以 2 元线性回归  $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$  为例。R 代码如下:

```
> x1<-c(9,223,83,21,193)
> x2<-c(98,52,185,263,76)
> c(237,415,479,630,438)->y
> lm(y~x1+x2)->mylm.lm
> summary(mylm.lm)
```

Call:

```
lm(formula = y ~ x1 + x2)
```

Residuals:

```
      1       2       3       4       5
2.462  2.037 -39.732 23.131 12.102
```

Coefficients:

```
          Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.9635    57.2566   0.191  0.8658
x1           1.2985     0.2341   5.547  0.0310 *
x2           2.1621     0.2622   8.246  0.0144 *
```

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Residual standard error: 33.69 on 2 degrees of freedom

Multiple R-squared: 0.9714, Adjusted R-squared: 0.9429

F-statistic: 34 on 2 and 2 DF, p-value: 0.02857

依次分析 `summary` 函数的结果, 进行显著性检验:

1) 分析 `Coefficients` 部分。 $\beta_0$  的预测值为 10.9635,  $\beta_1$  的预测值为 1.2985, 标准差为 0.2341,  $\beta_2$  的预测值为 2.1621, 标准差为 0.2622;  $\beta_1$  的  $t$  值为 5.547,  $\beta_2$  的  $t$  值为 8.246。  $\text{Pr(>|t|)}$  值即  $P$  值, 为  $T$  统计量的显著水平, 将该值与显著性水平比较, 可决定该自变量是否接受回归系数的假设检验, 该值越小越显著, 回归效果越好,  $\beta_1$  和  $\beta_2$  的  $\text{Pr(>|t|)}$  值均小于显著性水平 0.05, 但线性不显著, 后面的一个星号也说明了这一点, 自变量系数勉强通过显著性检验。



如果  $\Pr(>|t|)$  值  $>0.05$ , 表明该自变量的回归系数不显著, 这种情况下有必要考虑回归方程是否正确, 或者剔除一些不显著的变量, 重新建立更简单更精确的线性回归方程。

2) 分析 Residual standard error 部分, 该数值若较小则比较适合, 本例中残差的标准差为 33.69, 数值较大效果不理想。

3) 分析 Multiple R-squared 部分, 该部分为相关系数的平方, 该数值越大, 说明回归效果越理想。本例中相关系数的平方值为 0.9714。

4) 分析 F-statistic 部分的 p-value 值, 该值为 F 统计量的显著水平, 可用该值进行回归方程显著性检测, 数值越小越好, 越小则说明回归方程越显著。本例中的  $p$  值为 0.028 57, 回归效果一般。

此外, R 还提供了 influence.measures 进行回归诊断。下面建立了一个效果显著的回归模型:

```
> x1<-c(9,223,83,21,193)
> x2<-c(98,52,185,263,76)
> y<-x1+3*x2+runif(5,2,18)
> lm(y~x1+x2)->mylm.lm
> summary(mylm.lm)
```

Call:

```
lm(formula = y ~ x1 + x2)
```

Residuals:

```
1      2      3      4      5
0.03635 2.17997 1.16226 -0.40196 -2.97661
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	3.06344	4.67353	0.655	0.579476
x1	1.03200	0.01911	54.010	0.000343 ***
x2	3.02659	0.02140	141.412	5e-05 ***

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.75 on 2 degrees of freedom

Multiple R-squared: 0.9999, Adjusted R-squared: 0.9998

F-statistic: 1.163e+04 on 2 and 2 DF, p-value: 8.595e-05

观察以上结果,  $\Pr(>|t|)$  值和  $p$  值远小于 0.05, 说明回归方程和回归系数都非常显著, 回归效果非常好。

## 7.2.3 回归优化

通常可通过逐步回归来优化回归方程, 从可供选择的所有自变量中选出对  $Y$  有显著影响的变量建立方程, 忽略对  $Y$  无显著影响的变量。使用 R 的 step 函数可以完成逐步回归, 该函数以 AIC 信息统计量为准则, 选择最小的 AIC 信息统计量来删除自变量。

下面举例说明:

```
> x1<-c(9,223,83,21,193)
```

```
> x2<-c(98,52,185,263,76)
> x3<-c(25,35,48,58,42)
> y<-c(841,1570,2774,3923,2137)
> lm(y~x1+x2+x3)->mylm.lm
> summary(mylm.lm)
```

```
Call:
lm(formula = y ~ x1 + x2 + x3)
```

```
Residuals:
    1      2      3      4      5
8.075 39.072 -103.947  64.654 -7.854
```

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.395e+03  2.824e+02  -4.938   0.127
x1           4.873e-02  3.330e+00   0.015   0.991
x2           2.386e+00  6.309e+00   0.378   0.770
x3           7.973e+01  3.262e+01   2.444   0.247
```

```
Residual standard error: 129 on 1 degrees of freedom
Multiple R-squared:  0.997,    Adjusted R-squared:  0.988
F-statistic: 110.5 on 3 and 1 DF,  p-value: 0.06978
```

观察以上结果，从 `summary` 的分析结果可以看出，效果很不理想，`x1`、`x2`、`x3` 的系数 `P` 值全部大于 0.05，回归系数不显著，回归方程 `F` 分布的 `p` 值更不显著。

下面调用 `step` 函数进行优化。R 代码如下：

```
> step(mylm.lm)->mylm.step
Start: AIC=48.55
y ~ x1 + x2 + x3
```

	Df	Sum of Sq	RSS	AIC
- x1	1	4	16642	46.551
- x2	1	2380	19018	47.219
<none>			16638	48.550
- x3	1	99409	116048	56.262

```
Step: AIC=46.55
y ~ x2 + x3
```

	Df	Sum of Sq	RSS	AIC
<none>			16642	46.551
- x2	1	55362	72004	51.875
- x3	1	1388382	1405024	66.731

从以上 `step` 结果可以看出，选择去掉 `x1`，可以使 `AIC` 的值为最小值 46.551，因此在最终的回归方程中删除 `x1`。

```
> summary(mylm.step)
```

```
Call:
```

```
lm(formula = y ~ x2 + x3)

Residuals:
    1     2     3     4     5 
7.660 40.327 -103.459  64.579 -9.108 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1396.4131   180.3749   -7.742  0.01628 *
x2             2.2954     0.8899    2.579  0.12314
x3            80.1922     6.2082   12.917  0.00594 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 91.22 on 2 degrees of freedom
Multiple R-squared:  0.997,    Adjusted R-squared:  0.994 
F-statistic: 331.5 on 2 and 2 DF,  p-value: 0.003007
```

观察以上结果,从 summary 函数对优化后的回归方程的分析结果可以看出,相对没优化前,Residual standard error 值减少为 91.22,  $x_2$ 、 $x_3$  系数的 P 值减少为 0.123 14、0.005 94,  $x_3$  的系数通过检测,  $x_1$  因为不显著最终没有加入回归方程中, F 分布的 p-value 值也减少为 0.003 007, 总体来说优化后回归方程和回归系数更加显著, 回归效果更好。

## 7.2.4 主成分回归

主成分分析将多个指标中少数几个综合指标,通过降维技术,将多个变量简化成少数几个主成分的方法,这些主成分能反映原始变量的绝大部分信息,表示为原始变量的线性组合。以网上商场的购物满意度为例进行回归,表 7-9 是某网上商场的购物满意度指标。

表 7-9 某网上商场的购物满意度指标

网友评论次数 (X1)	好评次数 (X2)	商品浏览人数 (X3)	商品已上市天数 (X4)	购物满意度 (1-5)
200	120	800	210	4
600	480	3000	320	5
60	20	300	50	2
400	320	2200	260	4
150	80	450	130	3

### 1. 主成分分析

首先,使用 R 语言的 princomp 函数进行主成分分析:

```
> buy<-data.frame(
+ X1<-c(200,600,60,400,150),
+ X2<-c(120,480,20,320,80),
+ X3<-c(800,3000,300,2200,450),
+ X4<-c(210,320,50,260,130),
+ Y<-c(4,5,2,4,3)
+ )
```

```
> princomp(~X1+X2+X3+X4,cor=TRUE)->mypr
> summary(mypr,loadings=TRUE)
Importance of components:

      Comp.1      Comp.2      Comp.3      Comp.4
Standard deviation  1.9711809 0.32537035 0.092379136 6.793839e-03
Proportion of Variance 0.9713885 0.02646647 0.002133476 1.153906e-05
Cumulative Proportion 0.9713885 0.99785498 0.999988461 1.000000e+00

Loadings:
      Comp.1 Comp.2 Comp.3 Comp.4
X1 -0.506 -0.164  0.644  0.550
X2 -0.505 -0.289  0.203 -0.788
X3 -0.502 -0.381 -0.726  0.274
X4 -0.487  0.863 -0.130
```

观察以上结果，summary 函数输出了主成分分析信息，该信息主要包括以下内容：

- Standard deviation 行表示的是主成分的标准差（主成分的方差的开方）。
- Proportion of Variance 行表示的是方差的贡献率。
- Cumulative Proportion 行表示的是方差的累积贡献率。
- Loadings 栏表示降维后主成分对应于原始变量 X1、X2、X3、X4 的系数。在调用 summary 函数时，可以将参数 loadings 设为 TRUE，才能输出 loadings 信息。

## 2. 降维

该网上商场的满意度指标有网友评论次数 (X1)、好评次数 (X2)、商品浏览人数 (X3) 和商品已上市天数 (X4) 共 4 个，其中第 1 个主成分 Comp.1 的贡献率为 0.971 388 5，第 2 个主成分 Comp.2 的贡献率为 0.026 466 47，这两个主成分的累积贡献率已经达到 99.79%，并且后面两个主成分的方差贡献率很小，因此，可将后面两个主成分去掉，实现降维。

此外，还可做出碎石图（如图 7-1 所示），可直观地看出 X1 和 X2 的贡献率较大。R 代码如下：

```
screplot(mypr,type="lines")
```

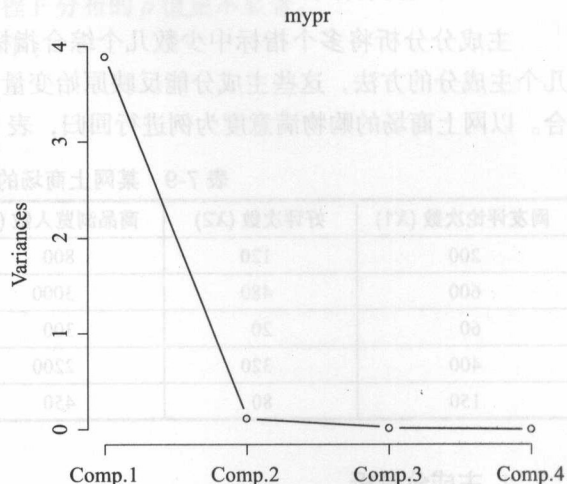


图 7-1 网上商场满意度指标碎石图

再观察对满意度指标分析结果中的 loadings 栏，可以将前两个主成分写成以下形式：

$$Z_1^* = -0.506X_1^* - 0.505X_2^* - 0.502X_3^* - 0.487X_4^*$$

$$Z_2^* = -0.164X_1^* - 0.289X_2^* - 0.381X_3^* + 0.863X_4^*$$

## 3. 回归模型建立

接着，可以建立这几个指标与满意度的回归模型，根据前面的主成分分析结果，只选择

前两个贡献率高的主成分  $X_1$  与  $X_2$ , 建立回归模型, 消除变量的多重共线性。

```
> pre<-predict(mypr)
> pre[,1]->buy$Z1# 第一主成分
> pre[,2]->buy$Z2# 第二主成分
> lm(Y~Z1+Z2,data=buy)->mylm# 建立回归模型
> summary(mylm)
```

Call:

```
lm(formula = Y ~ Z1 + Z2, data = buy)
```

Residuals:

```
1          2          3          4          5
0.06267  0.15488  0.01870 -0.25828  0.02203
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	3.60000	0.09770	36.847	0.000736 ***
Z1	-0.47551	0.04956	-9.594	0.010691 *
Z2	1.15957	0.30028	3.862	0.060988 .

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2185 on 2 degrees of freedom

Multiple R-squared: 0.9816, Adjusted R-squared: 0.9633

F-statistic: 53.48 on 2 and 2 DF, p-value: 0.01836

分析上述 summary 函数的执行结果, 可得到如下回归方程:

$$Y = 3.6 - 0.47551 \times Z_1 + 1.15957 \times Z_2$$

#### 4. 主成分变量变换

最后, 可将主成分变量进行变换:

```
> coef(mylm)->beta# 回归系数
> loadings(mypr)->myloading# 载荷
> mypr$center->mybar# 均值
> mypr$scale->mysd# 标准差
> (beta[2]*myloading[,1]+beta[3]*myloading[,2])/mysd->mycoef
> beta[1]-sum(mybar*mycoef)->beta0
> c(beta0,mycoef)
(Intercept)          X1          X2          X3          X4
1.3870124896  0.0002572658 -0.0005545049 -0.0001905035  0.0129419446
```

该网上商场的购物满意度的最终回归模型如下:

$$Y = 1.3870124896 + 0.0002572658 \times X_1 - 0.0005545049 \times X_2 - 0.0001905035 \times X_3 + 0.0129419446 \times X_4$$

### 7.2.5 广义线性模型

#### 1. 广义线性模型概述

广义线性模型 (Generalized Linear Model, GLM) 是一种被广泛应用的线性回归模式,



它是线性模型的扩展，其特点是不强行改变数据的自然度量，因为变量的总体均值可通过非线性或线性连接函数依赖于线性预测值，建立起可解释随机变量相关性的函数。

GLM 是简单最小二乘回归的扩展，主要是通过连接函数，建立响应变量的数学期望值与自变量线性组合的预测变量之间的关系。假设每个样本的观测值  $Y$  来自某个指数族分布，该分布的平均数  $\mu$  可由与该点独立的  $X$  来解释：

$$E(y)=\mu=g^{-1}(X\beta)$$

上式中， $E(y)$  为  $y$  的期望值， $X\beta$  是由未知待估计参数  $\beta$  与已知自变量  $X$  构成的线性估计式， $g$  为连接函数， $g^{-1}$  则为连接函数反函数。表 7-10 是经典的连接函数及其反函数（也可称为均值函数）。

表 7-10 经典的连接函数及其反函数

分布	名称	连接函数	均值函数
正态	恒等	$X\beta=\mu$	$\mu=X\beta$
指数	倒数	$X\beta=\mu^{-1}$	$\mu=(X\beta)^{-1}$
Gamma			
逆高斯	二次倒数	$X\beta=\mu^{-2}$	$\mu=(X\beta)^{-1/2}$
泊松	自然对数	$X\beta=\ln(\mu)$	$\mu=\exp(X\beta)$
二项	Logit	$X\beta=\ln\left(\frac{\mu}{1-\mu}\right)$	$\mu=\frac{\exp(X\beta)}{1+\exp(X\beta)}$
多项式			

简而言之，GLM 不但适用于连续数据，还适用于诸如属性数据、计数数据等离散数据，它通过连接函数将因变量的期望值与线性自变量联系，自变量的线性预测值是因变量的估计值，并对误差的分布给出误差函数，在 GLM 模型中，需要指定分布类型和连接函数。

在 R 中通常使用 GLM 函数构造广义线性模型，其中分布参数包括 binomial（二项分布）、gaussian（正态分布）、gamma（伽马分布）、poisson（泊松分布）等。

2. 广义线性模型实例

下面以二项分布（binomial）的 logistic 回归模型为例进行讲解。假设某网上商场试图建立某类商品的购买率（购买量 / 商品页面浏览次数）与气候之间的关系模型，这样就可根据当地气候的预测情况，推断该类商品的购买率，从而向消费者推荐该类商品。样本数据如表 7-11 所示。

表 7-11 雨量与购买率

温度	雨量（小为 0、中为 1、大为 2）	购买率
22	0	0.15
38	0	0.27
12	0	0.11
33	0	0.20
6	0	0.09

(续)

温度	雨量 (小为 0、中为 1、大为 2)	购买率
9	1	0.18
29	1	0.32
17	1	0.22
33	1	0.35
31	2	0.69
35	2	0.78
19	2	0.39
13	2	0.29
8	2	0.25

首先，建立广义线性模型。R 代码如下所示：

```
> mysale<-data.frame(
  temperature<-c(22,38,12,33,6,9,29,17,33,31,35,19,13,8),
  rainfall<-c(0,0,0,0,0,1,1,1,1,2,2,2,2,2),
  n<-rep(100,14),
  buy<-c(15,27,11,20,9,18,32,22,35,69,78,39,29,25)
)
> mysale$y<-cbind(mysale$buy,mysale$n-mysale$buy)
> glm(y~temperature+rainfall,family=binomial,data=mysale)->myglm
> summary(myglm)
```

Call:

```
glm(formula = y ~ temperature + rainfall, family = binomial,
    data = mysale)
```

Deviance Residuals:

```
      Min       1Q   Median       3Q      Max
-1.5231  -0.8861  -0.1604   1.1531   2.3457
```

Coefficients:

```
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.23592     0.20961 -15.438  <2e-16 ***
temperature  0.06136     0.00633   9.694  <2e-16 ***
rainfall     0.90678     0.08067  11.240  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

(Dispersion parameter for binomial family taken to be 1)

```
Null deviance: 243.940  on 13  degrees of freedom
Residual deviance: 19.553  on 11  degrees of freedom
AIC: 90.86
```

Number of Fisher Scoring iterations: 4

观察以上分析结果可以发现，回归参数为  $\beta_0=-3.235\ 92$ 、 $\beta_1=0.061\ 36$ 、 $\beta_2=-0.906\ 78$ ，系数参数通过了显著性水平  $\alpha=0.05$  的检验，回归模型如下：

$$P=\frac{\exp(-3.235\ 92+0.061\ 36\times\text{temperature}+0.906\ 78\times\text{rainfall})}{1+\exp(-3.235\ 92+0.061\ 36\times\text{temperature}+0.906\ 78\times\text{rainfall})}$$

其中 temperature 是温度，rainfall 是雨量。

接着，可根据该回归模型进行预测。比如温度为 30℃，大雨时，该类商品的购买率为多少？

```
> predict(myglm,data.frame(temperature=30,rainfall=2))>mypre
> mypre
1
0.41857
> exp(mypre)/(1+exp(mypre))>mybuy
> mybuy
1
0.603141
```

观察以上结果，最后一行表明，大雨且温度为 30℃时，购买率估计为 60.3%。再来看一个例子，表 7-12 是网上商场某类商品的若干名回头客的数据。

表 7-12 某类商品的若干名回头客的数据

商品打折	赠品数量 (0-2)	购买与否 (0 表示不购买, 1 表示购买)
0.82	0	0
0.92	0	0
0.68	0	0
0.7	0	0
0.62	0	1
0.52	0	1
0.42	0	1
0.85	1	0
0.95	1	0
0.65	1	1
0.75	1	0
0.7	1	1
0.5	1	1
0.38	1	1
0.85	2	1
0.95	2	1
0.65	2	1
0.75	2	1
0.6	2	1

(续)

商品打折	赠品数量 (0-2)	购买与否 (0 表示不购买, 1 表示购买)
0.48	2	1
0.98	2	0
0.3	2	1

对表 7-12 的数据运用 GLM 进行分析, 得到购买率 (Y) 和商品打折 (X1)、赠品数量 (X2) 的回归模型。

首先, 建立广义线性模型, R 代码如下:

```
mysale<-data.frame(
  X1=c(0.82,0.92,0.68,0.7,0.62,0.52,0.42,0.85,0.95,0.65,0.75,0.7,0.5,0.38,0.85,
  0.95,0.65,0.75,0.6,0.48,0.98,0.3),
  X2=rep(c(0,1,2),c(7,7,8)),
  Y=c(0,0,0,0,1,1,1,0,0,1,0,1,1,1,1,1,1,1,1,0,1))
>
> glm(Y~X1+X2,family=binomial,data=mysale)->myglm
> summary(myglm)
```

Call:

```
glm(formula = Y ~ X1 + X2, family = binomial, data = mysale)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-1.72890	-0.04535	0.00108	0.08745	1.28906

Coefficients:

	Estimate	Std. Error	z value	Pr(> z )
(Intercept)	24.388	14.086	1.731	0.0834 .
X1	-39.360	22.348	-1.761	0.0782 .
X2	6.373	3.751	1.699	0.0893 .

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 28.841 on 21 degrees of freedom

Residual deviance: 7.054 on 19 degrees of freedom

AIC: 13.054

Number of Fisher Scoring iterations: 8

>

观察以上分析结果可以看出, 回归参数为  $\beta_0=24.388$ 、 $\beta_1=-39.360$ 、 $\beta_2=6.373$ , 系数参数通过了显著性水平  $\alpha=0.1$  的检验, 回归模型如下:

$$P = \frac{\exp(24.388 - 39.360 \times X_1 + 6.373 \times X_2)}{1 + \exp(24.388 - 39.360 \times X_1 + 6.373 \times X_2)}$$

其中 X1 是商品打折, X2 是赠品数量。

然后, 根据该回归模型进行预测, 分别从以下几种情况进行预测。

1) 在商品打折 73%、赠品数量是 1 的情况下, 该类商品的购买率为 88.37%。R 代码如下:

```
> predict(myglm, data.frame(X1=0.73, X2=1)) -> mypre
> exp(mypre) / (1+exp(mypre)) -> mybuy
> mybuy
      1
0.8836742
>
```

2) 在商品打 85 折、赠品数量是 1 的情况下, 该类商品很可能不会被回头客购买。R 代码如下:

```
> predict(myglm, data.frame(X1=0.85, X2=1)) -> mypre
> exp(mypre) / (1+exp(mypre)) -> mybuy
> mybuy
      1
0.06323925
```

3) 在商品打 85 折、赠品数量是 2 的情况下, 该类商品非常有可能被回头客购买。R 代码如下:

```
> predict(myglm, data.frame(X1=0.85, X2=2)) -> mypre
> exp(mypre) / (1+exp(mypre)) -> mybuy
> mybuy
      1
0.9753328
>
```

## 7.3 小结

本章首先讲述了假设检验, 通过设立零假设 (想检验其正确性) 与对立假设 (通常反映研究者对未知参数的看法, 相对于零假设的其他论述), 研究某事件发生的可能性, 如果可能发生的话, 分析其预测区间值, 具体介绍了二项分布假设检验、数据分布检验、正态总体均值检验、列联表、符号检测、秩相关检验、Kendall 相关检验等。然后, 讲解了回归模型的若干问题, 回归通过规定因变量和自变量的关系来确定变量之间的因果关系, 具体介绍了回归预测、回归显著性检验、回归诊断与优化、主成分回归与广义线性回归等。

## 思考题

(1) 对某服务器进行稳定性检测, 记录 8 次无故障稳定工作的小时数, 分别为: 210、190、320、230、160、67、400、340, 经估计它符合  $\lambda=1/280$  的指数分布, 验证一下稳定性的分布。



(2) 网上商场某类商品经常发生投诉问题, 以往发生投诉的概率为 35%, 对这类商品的供应商进行规范和约束调整后, 随机抽取了 400 多个客户, 其中有 40 个投诉, 策略调整是否有效果?

(3) 对下面的 X 和 Y 进行 Kendall 相关检验, 检测其相关性。

```
> X<-c(150,210,90,190,230,211)
> Y<-c(40, 20,30,18,28,22)
```

(4) 对因变量  $y$  和自变量  $x_1$ 、 $x_2$ 、 $x_3$  建立线性回归模型, 并进行优化。

```
> x1<-c(9,223,83,21,193)
> x2<-c(108,52,180,259,82)
> x3<-c(8,5,10, 9, 2)
> c(237,415,479,630,438)->y
```

第  
...  
机

● 本基... 网... 时...

——林希元

自古圣贤之言学也，咸以躬行实践为  
先，识见言论次之。

——林希元

# 机器学习算法

学习就是在不断重复的工作中使自己的能力不断增强或改进，在下次执行相同或类似的任务时，会比原来做得更好或效率更高。机器学习研究指的是用计算机来模拟或实现人类学习活动，研究如何使机器通过识别和利用现有的知识来获取新知识和新技能。机器学习的内部表现为从未知到已知这样一个知识增长过程，其外部表现为某些性能和适应性得到系统的改善。机器学习形成了一套算法理论，这些算法使系统能完成原来不能完成的任务，或者能更好地完成原来可以完成的任务。

## 8.1 神经网络

### 1. 神经网络基本原理

人脑由上千亿条神经组成，每条神经平均又会连接到几千条其他的神经，通过这种连接方式，神经可以收发不同数量的能量。神经一个非常重要的功能就是，它们对能量的接收并不是立即作出响应，而是将它们累加起来，当这个累加的总和达到某个临界阈值时，它们才将自己的那部分能量发送给其他的神经。大脑通过调节这些连接的数目和强度来进行学习。

如图 8-1 所示是一个神经元的模型，神经网络复杂多样，由大量与该图类似的神经元及突触组成。神经科学界的共识是，人类大脑包含 1000 亿个神经元，每个神经元有 1 万个突触，数量巨大，组合方式复杂，联系广泛。也就是说，突触传递的机制复杂。

现在已经发现和阐明的突触传递机制有：突触后兴奋、突触后抑制、突触前抑制、突触前兴奋，以及远程抑制等。在突触传递机制中，释放神经递质是实现突触传递机能的中心环节，而不同的神经递质有着不同的作用性质和特点。

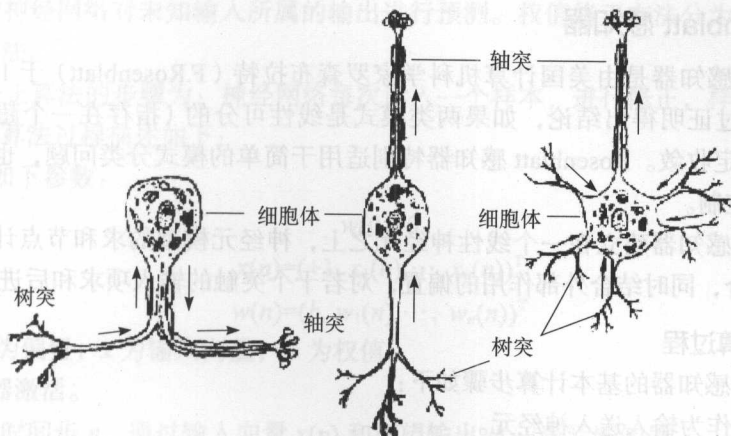


图 8-1 大脑神经元

人工神经网络 (ANN) 是一种模仿生物神经网络结构和功能的数学模型, 它使用大量的人工神经元连接来进行计算, 该网络由大量的“神经元”相互连接构成, 每个“神经元”代表一种特定的输出函数。又称为激励函数。每两个“神经元”间的连接代表一个通过该连接信号的加权值, 称之为权重, 这相当于人工神经网络的记忆。网络的输出则根据网络的连接规则来确定, 输出因权重值和激励函数的不同而不同。人工神经网络可理解为对自然界某种算法或者函数的逼近。

如图 8-2 所示是一个简单的人工神经元示意图。其中,  $x_i(t)$  等数据为这个神经元的输入, 代表其他神经元或外界对该神经元的输入;  $w_{ij}$  等数据为这个神经元的权重,  $u_i = \sum_j w_{ij} \cdot x_j(t)$  是对输入的求和, 为诱导局部域;  $y_i = f(u_i(t))$  为输出函数, 也称激励函数, 是对诱导局部域的再加工, 也是最终的输出。

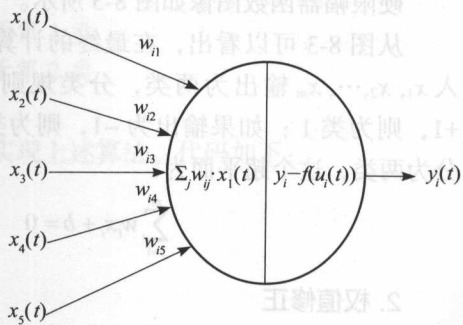


图 8-2 人工神经元

## 2. 神经网络发展历史

20 世纪 40 年代, 心理学家 McCulloch 和数学家 Pitts 联合提出了兴奋与抑制型神经元模型, 另外, Hebb 提出了神经元连接强度的修改规则; 到 20 世纪五六十年代, 该领域具有代表性的工作是 Rosenblatt 的感知机和 Widrow 的自适应性元件 Adaline; 1969 年, Minsky 和 Papert 合作出版了颇有影响力的著作《感知器》, 书中暗示感知器具有严重局限, 得出了消极悲观的论点, 使感知器与连接主义遭到冷落, 而感知器是神经网络的一种重要形式。在 20 世纪 70 年代, 人工神经网络的研究处于低潮, 20 世纪 70 年代末, 传统的冯·诺依曼 (Von Neumann) 数字计算机在模拟视听觉的人工智能方面遇到了物理上不可逾越的极限。但与此同时, Rumelhart、McClelland 和 Hopfield 等人在神经网络领域取得了突破性进展, 神经网络的热潮再次掀起。

### 8.1.1 Rosenblatt 感知器

Rosenblatt 感知器是由美国计算机科学家罗森布拉特 (F.Rosenblatt) 于 1957 年提出的。F.Rosenblatt 经过证明得出结论, 如果两类模式是线性可分的 (指存在一个超平面将它们分开), 则算法一定收敛。Rosenblatt 感知器特别适用于简单的模式分类问题, 也可用于基于模式分类的学习控制。

Rosenblatt 感知器建立在一个线性神经元之上, 神经元模型的求和节点计算作用于突触输入的线性组合, 同时结合外部作用的偏置, 对若干个突触的输入项求和后进行调节。

#### 1. 基本计算过程

Rosenblatt 感知器的基本计算步骤如下:

1) 将数据作为输入送入神经元。

2) 通过权值和输入共同计算诱导局部域, 诱导局部域是指求和节点计算得到的结果, 计算公式如下:

$$v = \sum_{i=1}^m w_i x_i + b$$

3) 以硬限幅器为输出函数, 诱导局部域被送入硬限幅器, 形成最终的输出硬限幅器的工作原理如下。

硬限幅器输入为正时, 神经元输出 +1, 反之输出 -1。计算公式为:

$$f(v) = \begin{cases} 1, & v > 0 \\ -1, & v \leq 0 \end{cases}$$

硬限幅器函数图像如图 8-3 所示。

从图 8-3 可以看出, 在最终的计算结果中, 把外部的输入  $x_1, x_2, \dots, x_m$  输出为两类, 分类规则是: 如果输出函数是 +1, 则为类 1; 如果输出为 -1, 则为类 2。感知器被超平面分为两类, 这个超平面为:

$$\sum_{i=1}^m w_i x_i + b = 0$$

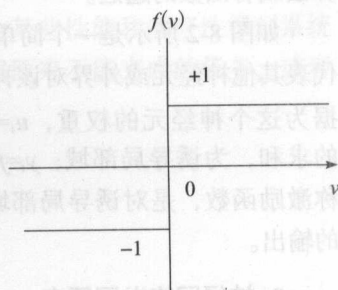


图 8-3 硬限幅器函数图像

#### 2. 权值修正

在上述计算过程中, 第二步涉及一个重要的参数, 即权值。如何确定权值才能保证输出能被正确地分类到 +1 和 -1 呢?

首先, 会产生一个初始权值, 由初始权值计算得到的输出结果肯定会有误差。接着, 要想办法让误差减少, 这个过程就是权值  $w$  修正的过程。

新的问题产生了, 哪些数据用来输入呢? 在用神经网络进行机器学习时, 输入数据是确定的, 但输出结果是未知的, 这些输出结果正是我们需要用神经网络预测的结果。解决这个问题的关键在于样本。样本是研究中实际观测或调查的一部分, 研究对象的全部称为总体, 样本必须能够正确反映总体情况。所以, 首先要用样本将神经网络训练好, 在确定权值后,



再用训练好的神经网络对未知输入所属的输出进行预测。权值修正方法分为单样本修正算法和批量修正算法。

单样本修正算法的步骤为：神经网络每次读入一个样本，进行修正，样本读取完毕，修正过程结束。算法过程描述如下：

1) 设置如下参数：

$$\begin{aligned} w(0) &= 0 \\ x(n) &= (+1, x_1(n), \dots, x_n(n))^T \\ w(n) &= (b, w_1(n), \dots, w_n(n))^T \end{aligned}$$

其中， $b$  为偏置， $x$  为输入向量， $w$  为权值。

2) 感知器激活。

对于每个时间步  $n$ ，通过输入向量  $x(n)$  和期望输出  $d(n)$  激活感知器。

3) 计算感知器的输出。

$$\begin{aligned} y(n) &= \text{sgn}(w^T(n)x(n)) \\ \text{sgn} &= \begin{cases} +1, & v > 0 \\ -1, & v \leq 0 \end{cases} \end{aligned}$$

其中， $n$  为时间步， $x(n)$  为输入向量， $w(n)$  为权值向量， $\text{sgn}$  为硬限幅函数， $v$  为硬限幅函数的输入值。

4) 更新感知器的权值向量。

$$\begin{aligned} w(n+1) &= w(n) + \eta(d(n) - y(n))x(n) \\ 0 &< \eta < 1 \end{aligned}$$

$$d(n) = \begin{cases} +1, & x(n) \text{ 属于第 1 类} \\ -1, & x(n) \text{ 属于第 2 类} \end{cases}$$

其中， $\eta$  为学习速率（调整更新的步伐）。

下面用神经网络学习逻辑或的运算，用 Python 实现上述算法。代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-1.py
import numpy as np
b=0
a=0.5
x = np.array([[0,1,1],[0,1,0],[0,0,0],[0,0,1]])
d = np.array([1,1,0,1])
w=np.array([b,0,0])
def sgn(v):
    if v>0:
        return 1
    else:
        return 0
def comy(myw,myx):
```

```

        return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    return oldw+a*(myd-comy(oldw,myx))*myx
i=0
for xn in x:
    w=neww(w,d[i],xn,a)
    i+=1
for xn in x:
    print "%d or %d => %d"%(xn[1],xn[2],comy(w,xn))

```

如下程序输出结果正确。

```

1 or 1 => 1
1 or 0 => 1
0 or 0 => 0
0 or 1 => 1

```

现在来测试一个更复杂的例子。针对下面两类函数训练神经网络，将一组  $(x, y)$  值划分为以下两类函数之一：

□  $2x+1=y$  为第 1 类。

□  $7x+1=y$  为第 2 类。

代码如下：

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#8-2.py
import numpy as np
b=1
a=0.3
x=np.array([[1,1,3],[1,2,5],[1,1,8],[1,2,15],[1,3,7],[1,4,29]])
d=np.array([1,1,-1,-1,1,-1])
w=np.array([b,0,0])
def sgn(v):
    if v>=0:
        return 1
    else:
        return -1
def comy(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    print comy(oldw,myx)
    return oldw+a*(myd-comy(oldw,myx))*myx
i=0
for xn in x:
    print xn
    w=neww(w,d[i],xn,a)
    i+=1
    print w
for xn in x:
    print "%d %d => %d"%(xn[1],xn[2],comy(w,xn))
test=np.array([b,9,19])

```

```
print "%d %d => %d" %(test[1],test[2],comy(w,test))
test=np.array([b,9,64])
print "%d %d => %d" %(test[1],test[2],comy(w,test))
```

在上面代码中,使用了[1,3]、[2,5]、[1,8]、[2,15]、[3,7]、[4,29]这几组数据对该神经网络进行训练。运行该程序,对从未在样本中出现过的数据[9,19]、[9,64]进行分类。通过前面的函数定义可以知道,[9,19]属于第1类,[9,64]属于第2类。如下所示:

```
9 19 => 1
9 64 => -1
```

接着输出训练之后的神经网络权值参数,代码如下:

```
>>> w
array([[ 1. , 1.2, -0.6])
```

根据上述参数,可以确定神经网络的分类线方程为:

$$1.2x - 0.6y + 1 = 0 \Rightarrow 1.2x + 1 = 0.6y \Rightarrow y \approx 2x + 1.68$$

那么对于不完全符合上述两个函数定义的数据,通过训练好的神经网络也能近似地划分到上述函数中。试着将前面的代码修改一下,加入两个不太规则的测试数据点[9,16]、[9,60],并把样本[b,2,5]改为不规则的[b,2,3],同时加入绘图命令,同时加入绘图命令,绘制数据点,然后使用上面计算的分类线方程绘制分类线(这样能直观地观察结果),如图8-4所示。

在如图8-4所示的样本数据点中,大的圆点属于第2类,输出为-1,星号属于第1类,输出为1;在测试数据点中,叉号属于第2类,输出为-1,小点属于第1类,输出为1。中间的虚线就是分类线,这条分类线把坐标系内的点分成为两类,分类线以上的点输出为-1,以下的点输出为1,训练好的神经网络通过这条分类线决定如何对输入所属的分类进行预测。

修改后的Python代码如下:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#8-3.py
import numpy as np
import pylab as pl
b=1
a=0.3
x=np.array([[1,1,3],[1,2,3],[1,1,8],[1,2,15],[1,3,7],[1,4,29]])
d=np.array([1,1,-1,-1,1,-1])
w=np.array([b,0,0])
def sgn(v):
    if v>=0:
```

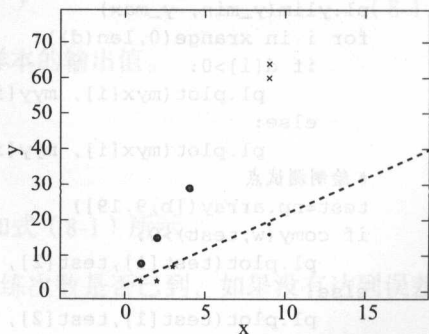


图8-4 神经网络分类(附彩图)

```

        return 1
    else:
        return -1

def comy(myw, myx):
    return sgn(np.dot(myw.T, myx))

def neww(oldw, myd, myx, a):
    return oldw + a * (myd - comy(oldw, myx)) * myx

i = 0
for xn in x:
    w = neww(w, d[i], xn, a)
    i += 1

myx = x[:, 1]
myy = x[:, 2]
pl.subplot(111)
x_max = np.max(myx) + 15
x_min = np.min(myx) - 5
y_max = np.max(myy) + 50
y_min = np.min(myy) - 5
pl.xlabel(u"x")
pl.xlim(x_min, x_max)
pl.ylabel(u"y")
pl.ylim(y_min, y_max)
for i in xrange(0, len(d)):
    if d[i] > 0:
        pl.plot(myx[i], myy[i], 'r*')
    else:
        pl.plot(myx[i], myy[i], 'ro')
# 绘制测试点
test = np.array([b, 9, 19])
if comy(w, test) > 0:
    pl.plot(test[1], test[2], 'b.')
else:
    pl.plot(test[1], test[2], 'bx')
test = np.array([b, 9, 64])
if comy(w, test) > 0:
    pl.plot(test[1], test[2], 'b.')
else:
    pl.plot(test[1], test[2], 'bx')
test = np.array([b, 9, 16])
if comy(w, test) > 0:
    pl.plot(test[1], test[2], 'b.')
else:
    pl.plot(test[1], test[2], 'bx')
test = np.array([b, 9, 60])
if comy(w, test) > 0:
    pl.plot(test[1], test[2], 'b.')
else:
    pl.plot(test[1], test[2], 'bx')
# 绘制分类线
testx = np.array(range(0, 20))
testy = testx * 2 + 1.68
pl.plot(testx, testy, 'g--')
pl.show()

```

上面给出的是单样本感知器算法, 算法仅读取一次样本, 每读取一个样本, 就是一次迭代。每次迭代时, 只考虑了用一个训练模式修正权矢量。实际上, 可以将几个训练模式一起考虑, 而批量修正算法则考虑了使用代价函数来进行分类误差率的控制。批量修正算法要对样本进行多次读取, 直到神经网络的误差率降到合适的程度才停止样本的训练, 这是它与单样本修正算法本质的区别。算法中的误差率使用最直观的被错分类的样本数量准则。

批量修正算法的核心在于其权值更新策略。批量修正算法采用的是梯度下降原理, 其计算公式为:

$$w(k+1)=w(k)-\eta(k) \nabla J(w(k))$$

其中,  $\eta(k)$  称为学习率,  $\nabla J(w(k))$  为梯度, 计算方法为:

$$\nabla J(w(k))=\sum_{y \in Y_k}(-y)$$

其中,

$Y_k$  = 被错分类的样本输出值集合

最终得出权值更新策略为:

$$w(k+1)=w(k)+\eta(k) \sum_{y \in Y_k} d^* y \quad (8-1)$$

在上式中,  $d$  为样本实际输出值,  $y$  为被错分类的样本的输出值。

具体算法过程如下:

1) 初始化权值、学习率, 以及期望误差率。

2) 读取所有样本数据。

3) 依次对样本进行训练, 更新权值, 其更新策略如式 (8-1) 所示。

4) 检查  $\left| \eta(k) \sum_{y \in Y_k} d^* y \right|$  是否小于指定误差率, 或者训练次数是否已到, 如果没有达到误差率的要求且训练次数未到, 转到第 2 步执行。

继续使用单样本修正法的样本, 将一组  $(x, y)$  的输入划分为以下两类函数之一:

□  $2x+1=y$  为第 1 类。

□  $7x+1=y$  为第 2 类。

下面用 Python 实现这个神经网络, 其中加入了一个不规则的样本数据 [1,2,3], 最后用 [9,19] 和 [3,22] 对训练成功后的网络进行测试。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-4.py
import numpy as np
b=1
a=0.5
x = np.array([[1,1,3],[1,2,3],[1,1,8],[1,2,15]])
d =np.array([1,1,-1,-1])
w=np.array([b,0,0])
wucha=0
```



```

ddcount=50
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def comy(myw,myx):
    return sgn(np.dot(myw.T,myx))
def tiduxz(myw,myx,mya):
    i=0
    sum_x=np.array([0,0,0])
    for xn in myx:
        if comy(myw,xn)!=d[i]:
            sum_x+=d[i]*xn
        i+=1
    return mya*sum_x
i=0
while True:
    tdxz=tiduxz(w,x,a)
    w=w+tdxz
    i=i+1
    if abs(tdxz.sum())<=wucha or i>=ddcount:break
test=np.array([1,9,19])
print "%d %d => %d"%(test[1],test[2],comy(w,test))
test=np.array([1,3,22])
print "%d %d => %d"%(test[1],test[2],comy(w,test))

```

运行程序后可发现,训练效果很好,测试数据被正确分类。

```

9 19 => 1
3 22 => -1

```

### 3. LMS 算法

LMS 算法全称为 least mean square 算法,中文是最小均方算法。在 ANN 领域内,均方误差是指样本预测输出值与实际输出值之差平方的期望值,记为 *MSE*。设 *observed* 为样本真值, *predicted* 为样本预测值,计算公式如下:

$$MSE = \frac{1}{n} \sum_{i=1}^n (observed_i - predicted_i)^2$$

LMS 算法的策略是使均方误差最小,该算法运行在一个线性神经元上,使用的是批量修正算法,其误差信号为:

$$e(n) = d(n) - \hat{w}(n)$$

然后在误差信号的基础上计算梯度向量,公式如下:

$$\frac{\partial e(n)}{\partial \hat{w}(n)} = -X(n)e(n)$$

最后,生成权值调整方案。公式如下:

$$\hat{w}(n+1) = \hat{w}(n) + \eta X(n)e(n)$$

在上式中,  $\eta$  表示学习率, 该值越小, LMS 算法执行得越精确, 但同时算法收敛速度越慢。下面使用 LMS 算法实现逻辑或运算, 将学习率设为 0.1。代码如下:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-5.py
import numpy as np
b=1
a=0.1
x = np.array([[1,1,1],[1,1,0],[1,0,1],[1,0,0]])
d =np.array([1,1,1,0])
w=np.array([b,0,0])
expect_e=0.005
maxtrycount=20
def sgn(v):
    if v>0:
        return 1
    else:
        return 0
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
mycount=0
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye/=float(i)
    mycount+=1
    print u" 第 %d 次调整后的权值: "%mycount
    print w
    print u" 误差: %f"%mye
    if mye<expect_e or mycount>maxtrycount:break
for xn in x:
    print "%d or %d => %d "%(xn[1],xn[2],get_v(w,xn))
```

下面是程序的运行结果, 可以看出, 通过 13 次迭代, 训练结束, 对测试值的输出正确。

```
.....
.....
第 12 次调整后的权值:
[-0.1  0.1  0.1]
误差: 0.500000
第 13 次调整后的权值:
[-0.1  0.1  0.1]
```

```
误差: 0.000000
```

```
1 or 1 => 1
```

```
1 or 0 => 1
```

```
0 or 1 => 1
```

```
0 or 0 => 0
```

```
>>>
```

另外，可应用 LMS 算法实现比逻辑与更复杂的算法，比如：在输入矩阵中，如果  $x$  向量的整除结果为 6，则表示为一类，输出为 1；若结果为 3 则是另一类，输出为 -1。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-6.py
import numpy as np
b=1
a=0.1
x = np.array([[1,1,6],[1,2,12],[1,3,9],[1,8,24]])
d = np.array([1,1,-1,-1])
w=np.array([b,0,0])
expect_e=0.005
maxtrycount=20
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
mycount=0
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye/=float(i)
    mycount+=1
    print u" 第 %d 次调整后的权值: "%mycount
    print w
    print u" 误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break
for xn in x:
    print "%d    %d => %d"%(xn[1],xn[2],get_v(w,xn))
test=np.array([1,9,27])
print "%d    %d => %d"%(test[1],test[2],get_v(w,test))
```

```
test=np.array([1,11,66])
print "%d %d => %d"%(test[1],test[2],get_v(w,test))
```

通过9次训练后，误差率为0，用样本数据和测试数据进行验证，准确无误。下面是执行结果：

```
.....
```

```
.....
第 8 次调整后的权值:
```

```
[ 1.4 -2.8  0.6]
```

```
误差: 3.0000000
```

```
第 9 次调整后的权值:
```

```
[ 1.4 -2.8  0.6]
```

```
误差: 0.0000000
```

```
1      6 => 1
```

```
2     12 => 1
```

```
3      9 => -1
```

```
8     24 => -1
```

```
9     27 => -1
```

```
11     66 => 1
```

#### 4. Rosenblatt 感知器的局限性

不是所有的数据都能被 Rosenblatt 感知器正确分类。比如说下面的样本数据：

```
x = np.array([[1,1,6],[1,3,12],[1,3,9],[1,3,21],[1,2,16],[1,3,15]]) d =np.
array([1,1,-1,-1,1,-1])
```

应用 LMS 算法对这些数据训练 200 次，效果仍非常差，样本数据和测试数据都无法正确分类。如下所示：

```
.....
```

```
.....
第 199 次调整后的权值:
```

```
[ 18.  -17.2  0.8]
```

```
误差: 2.666667
```

```
第 200 次调整后的权值:
```

```
[ 18.  -17.4  -0.8]
```

```
误差: 2.666667
```

```
第 201 次调整后的权值:
```

```
[ 18.4 -16.8  1.8]
```

```
误差: 2.666667
```

```
1      6 => 1
```

```
3     12 => -1
```

```
3      9 => -1
```

```
3     21 => 1
```

```
2     16 => 1
```

```
3     15 => -1
```

```
9     27 => -1
```

```
11     66 => -1
```

为什么会出现这种情况？试着修改 8-6.py 的样本数据，同时绘制包括这些点的散点图。

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-7.py
import numpy as np
import pylab as pl
b=1
a=0.1
x = np.array([[1,1,6],[1,3,12],[1,3,9],[1,3,21],[1,2,16],[1,3,15]])
d = np.array([1,1,-1,-1,1,-1])
w=np.array([b,0,0])
expect_e=0.005
maxtrycount=200
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
mycount=0
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye/=float(i)
    mycount+=1
    print u" 第 %d 次调整后的权值: "%mycount
    print w
    print u" 误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break
for xn in x:
    print "%d %d => %d"%(xn[1],xn[2],get_v(w,xn))
test=np.array([1,9,27])
print "%d %d => %d"%(test[1],test[2],get_v(w,test))
test=np.array([1,11,66])
print "%d %d => %d"%(test[1],test[2],get_v(w,test))
myx=x[:,1]
myy=x[:,2]
pl.subplot(111)
x_max=np.max(myx)+10
x_min=np.min(myx)-5
y_max=np.max(myy)+50
y_min=np.min(myy)-5
pl.xlabel(u"x")

```



```

pl.xlim(x_min, x_max)
pl.ylabel(u"y")
pl.ylim(y_min, y_max)
# 绘制样本点
for i in xrange(0, len(d)):
    if d[i]>0:
        pl.plot(myx[i], myy[i], 'r*')
    else:
        pl.plot(myx[i], myy[i], 'ro')
# 绘制测试点
test=np.array([1,9,27])
pl.plot(test[1],test[2], 'bx')
test=np.array([1,11,66])
pl.plot(test[1],test[2], 'bx')
pl.show()

```

在如图 8-5 所示的散点图中, 实心圆圈和星号是两类不同的样本点, 叉号为测试点。在这张图中, 无法画出一条线来将两类点分开, 只有曲线才能将它们分开, 但线性方程根本不可能产生的曲线, 因此 Rosenblatt 感知器不适用于非线性分类。

### 5. LMS 的学习率退火算法

模拟退火算法来源于固体退火原理。退火就是将材料加热后再经特定速率冷却, 目的是增大晶粒的体积, 并且减少晶格中的缺陷。材料中的原子原来会停留在使内能有局部最小值的位置, 加热使能量变大, 原子会离开原来位置, 而随机在其他位置中移动。退火冷却时速度较慢, 徐徐冷却时粒子渐趋有序, 原子有可能找到内能比原先更低的位置, 最后在常温时达到基态, 内能减为最小。

根据 Metropolis 准则, 粒子在温度  $T$  时趋于平衡的概率为  $e^{-\Delta E/(kT)}$ , 其中  $E$  为温度  $T$  时的内能,  $\Delta E$  为其改变量,  $k$  为 Boltzmann 常数。用固体退火模拟组合优化问题, 将内能  $E$  模拟为目标函数值  $f$ , 温度  $T$  演化成控制参数  $t$ , 即得到解组合优化问题的模拟退火算法: 由初始解  $i$  和控制参数初值  $t$  开始, 对当前解重复进行“产生新解→计算目标函数差→接受或舍弃”的迭代, 并逐步衰减  $t$  值, 算法终止时的当前解即为所得近似最优解, 这是基于蒙特卡罗迭代求解法的一种启发式随机搜索过程。退火过程由冷却进度表控制, 包括控制参数的初值  $t$  及其衰减因子  $\Delta t$ 、每个  $t$  值时的迭代次数  $L$  和停止条件  $S$ 。

针对学习率不变化、收敛速度较慢的情况, 即可使用退火算法方案, 让学习率随时间而变化。学习率计算公式为:

$$\eta(n) = \frac{\eta_0}{1 + (n/\tau)}$$

下面应用退火算法对代码 8-6.py 做一些改动, 代码如下:

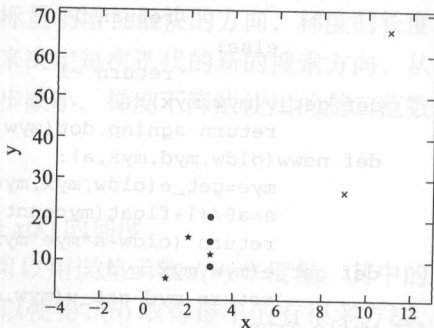


图 8-5 散点图

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-8.py
import numpy as np
import math
b=1
a0=0.1
a=0.0
r=5.0
x = np.array([[1,1,6],[1,2,12],[1,3,9],[1,8,24]])
d =np.array([1,1,-1,-1])
w=np.array([b,0,0])
expect_e=0.05
maxtrycount=20
mycount=0
def sgn(v):
    if v>0:
        return 1
    else:
        return -1
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    a=a0/(1+float(mycount)/r)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)
while True:
    mye=0
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye=math.sqrt(mye)
    mycount+=1
    print u" 第 %d 次调整后的权值: "%mycount
    print w
    print u" 误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break
    for xn in x:
        print "%d %d => %d"%(xn[1],xn[2],get_v(w,xn))
    test=np.array([1,9,27])
    print "%d %d => %d"%(test[1],test[2],get_v(w,test))
    test=np.array([1,11,66])
    print "%d %d => %d"%(test[1],test[2],get_v(w,test))

```

从程序运行结果来看，仅仅 7 次就完成了训练，训练次数大大减少。

.....  
.....

第 6 次调整后的权值:

```
[ 1.28888889 -1.55238095  0.29642857]
```

```
误差: 3.464102
```

```
第 7 次调整后的权值:
```

```
[ 1.28888889 -1.55238095  0.29642857]
```

```
误差: 0.000000
```

```
1    6 => 1
```

```
2   12 => 1
```

```
3    9 => -1
```

```
8   24 => -1
```

```
9   27 => -1
```

```
11  66 => 1
```

```
>>>
```

## 8.1.2 梯度下降

### 1. 梯度下降概述

上一节中曾提到梯度的概念，究竟什么是梯度和梯度下降呢？

梯度是一个向量场，标量场中某一点上的梯度指向标量场增长最快的方向，梯度的长度是这个最大的变化率。梯度下降，就是利用负梯度方向来决定每次迭代的新的搜索方向，从而使得在每次迭代过程中，都能让待优化的目标函数逐步减小。梯度下降法使用的是二范数下的最速下降法。最速下降法的一种简单形式如下：

$$x(k+1)=x(k)-a*g(k)$$

其中， $a$ 称为学习速率，可以是较小的常数。 $g(k)$ 是 $x(k)$ 的梯度。

机器学习算法效果究竟如何，或者说误差为多少，可以用误差函数 $J(\theta)$ 来度量。其中的参数 $\theta$ 在神经网络中可以理解为权值。如何调整权值 $\theta$ 以使得 $J(\theta)$ 取得最小值有很多方法，梯度下降法是按下面的步骤进行的：

- 1) 对 $\theta$ 赋值，这个值可以是随机的，也可以让 $\theta$ 是一个全零的向量。
- 2) 改变 $\theta$ 的值，使得 $J(\theta)$ 按梯度下降的方向进行减少。

为了更清楚地表达，先来看一下如图 8-6 所示的误差曲面及梯度下降图。

图 8-6 是表示参数 $\theta$ 与误差函数 $J(\theta)$ 关系的图，也称误差曲面图。该图上的方向表明，随着迭代次数的增加，误差走向了曲面的最小误差点。

红色较凸起的部分是表示 $J(\theta)$ 有着比较高的取值，对其进行神经网络训练时，最完美的目标是：让 $J(\theta)$ 的值尽量低，降到最低处，也就是最凹的部分。 $\theta_0$ 、 $\theta_1$ 表示 $\theta$ 向量的两个维度。

梯度下降法的第一步是给 $\theta$ 一个初值，假设随机给的初值是在最高的十字点处；然后将 $\theta$ 按照梯度下降的方向进行调整，就会使得 $J(\theta)$ 往更低的方向变化，如图 8-7 所示。算法的结束将是在 $\theta$ 下降到无法继续下降为止。当然，可能梯度下降的最终点并非是全局最小点（图 8-6 的路径中的最后一个点），而是一个局部最小点（图 8-7 的路径中的最后一个点），比如图 8-7 所示的情况，而图 8-6 中的曲线路径则是完美的梯度下降过程。

图 8-7 中所示的这种情况是在神经网络训练中要尽量避免出现的，这里的梯度下降到局部最小点后停止，神经网络在一个不正确的位置收敛了，训练停止，这时即使继续训练也没有任何效果。

Gradient Descent Algorithm

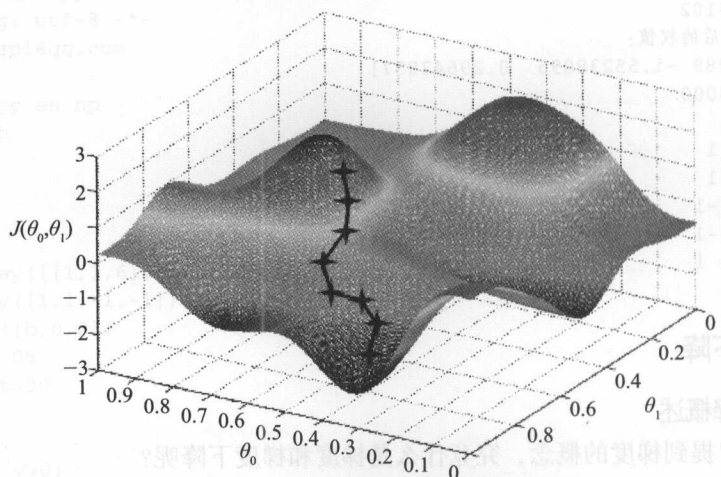


图 8-6 误差曲面及梯度下降 (附彩图)

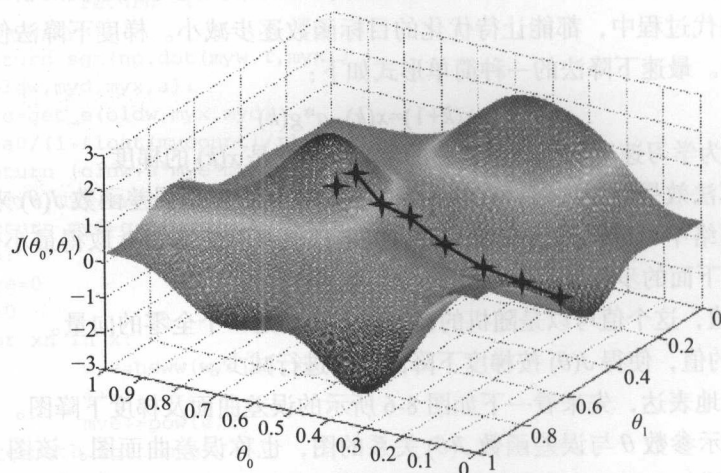


图 8-7 落到局部最小点的梯度下降 (附彩图)

## 2. 梯度下降法涉及的数学知识

1) 导数的几何意义: 导数的几何意义在于, 函数在某一点的导数等于它的图像上这一点处之切线的斜率, 如图 8-8 所示。

其斜率为:

$$\tan \alpha = \lim_{\Delta x \rightarrow 0} \tan \varphi = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

2) 积分的几何意义: 对于一个给定的正实值

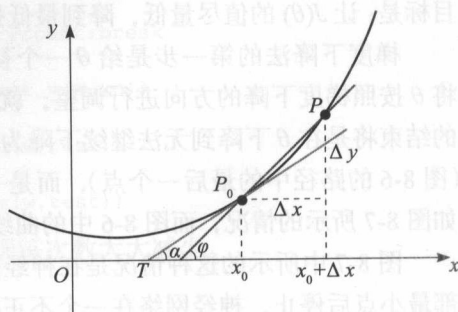


图 8-8 导数

函数  $f(x)$ ,  $f(x)$  在一个实数区间  $[a, b]$  上的定积分为:

$$\int_a^b f(x) dx$$

定积分的几何意义可以理解为在  $Oxy$  坐标平面上, 由曲线  $(x, f(x))$ 、直线  $x=a$ ,  $x=b$  以及  $x$  轴围成的曲边梯形的面积, 这个面积可以是一块面积, 也可以是多块面积的和 (注意, 这里的面积有正负之分)。

3) 微分的几何意义: 一元函数的微分与自变量的微分之商等于该函数的导数, 所以导数也叫做微商。

设  $\Delta x$  是曲线  $y=f(x)$  上的点  $P$  在横坐标上的增量,  $\Delta y$  是曲线在点  $P$  处相对于  $\Delta x$  的纵坐标增量,  $dy$  是曲线在点  $P$  的切线对应  $\Delta x$  在纵坐标上的增量, 如图 8-9 所示。

4) 偏导数: 拥有多个自变量的函数的偏导数是指在保持其他自变量恒定的情况下, 该函数相对于其中某个自变量的导数。函数  $f$  关于变量  $x$  的偏导数写为  $f'_x$  或  $\frac{\partial f}{\partial x}$ 。

$f$  是一个多元函数。例如:

$$f(x, y) = x^2 + xy + y^2$$

因为曲线上的每一点都有无穷多条切线, 描述这种函数的导数相当困难。偏导数就是选择其中一条切线, 并求出它的斜率。

多自变量函数  $f(x_1, \dots, x_n)$  在点  $(a_1, \dots, a_n)$  处, 对于某个自变量  $x_i$  的偏导数, 其定义为:

$$\frac{\partial f}{\partial x_i}(a_1, \dots, a_n) = \lim_{h \rightarrow 0} \frac{f(a_1, \dots, a_i + h, \dots, a_n) - f(a_1, \dots, a_n)}{h}$$

5) 梯度: 欧几里得空间  $R^n$  中的函数  $f(x_1, \dots, x_n)$  对每个自变量  $x_j$  具有偏导数  $\partial f / \partial x_j$ , 在点  $a$  处, 这些偏导数定义了一个向量:

$$\nabla f(a) = \left( \frac{\partial f}{\partial x_1}(a), \dots, \frac{\partial f}{\partial x_n}(a) \right)$$

这个向量称为  $f$  在点  $a$  处的梯度。

### 3. 梯度下降的原理

如果实值函数  $F(x)$  在点  $a$  处可微且有定义, 那么该函数在  $a$  点沿着梯度相反的方向下降最快, 从函数  $F$  对局部极小值的初始估计值  $x_0$  出发, 存在如下序列  $x_0, x_1, x_2, \dots$ , 使得

$$x_{n+1} = x_n - \gamma_n \nabla F(x_n), n \geq 0$$

因此可得到

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \dots$$

如果顺利的话, 序列  $(X_n)$  将收敛到期望的极值。

### 4. 梯度下降和 delta 法则

delta 法则克服了感应器法则的不足, 在线性不可分的训练样本上, 可以有效地收敛, 并

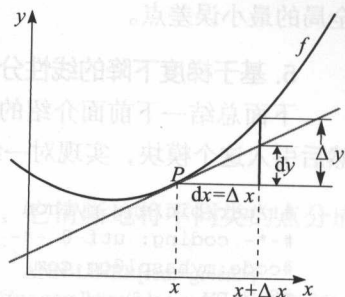


图 8-9 微分



接近目标的最佳近似值。**delta** 法则的关键思想是：使用梯度下降来搜索可能的权向量假设空间，以找到最佳拟合训练样例的权向量。

**delta** 法则为反向传播算法提供了基础，而反向传播算法能够学习多个单元的互连网络，在包含多种不同类型的连续参数化假设的空间中，梯度下降是必须遍历这样的空间的所有算法的基础。

误差曲面是一个抛物面，存在一个单一全局最小值，梯度下降搜索从一个任意的初始权向量开始，然后沿误差曲面最陡峭下降的方向，以很小的步伐反复修改这个向量，直到得到全局的最小误差点。

## 5. 基于梯度下降的线性分类器

下面总结一下前面介绍的线性神经网络代码，并将它们整理成 Python 模块 `mplannliner`，然后引入这个模块，实现对一组输入的线性分类。以下程序有详细的注释说明。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-9.py
import mplannliner as nplann
traindata1=[[9,25],-1],[[5,8],-1],[[15,31],-1],[[35,62],-1],[[19,40],-1],[[28,6
5],1],[[20,59],1],[[9,41],1],[[12,60],1],[[2,37],1]]
myann=nplann.Mplannliner()
# 样本初始化
myann.samples_init(traindata1)
# 学习率初始化
myann.a_init(0.1)
# 搜索时间常数初始化
myann.r_init(50)
# 最大训练次数
myann.maxtry_init(500)
# 期望最小误差
myann.e_init(0.05)
# 训练
myann.train()
# 仿真，测试，对未知样本分类
myc=myann.simulate([35,68])
print "[35,68]"
if myc==1:
    print u" 正类 "
else:
    print u" 负类 "
# 将测试点在最终效果图上显示出来，将它加入 drawpoint 集，测试点表现为 "*"，并且色彩由其最终的
# 分类结果而决定
myann.drawpoint_add([35,68])
myc=myann.simulate([35,82])
print "[35,82]"
if myc==1:
    print u" 正类 "
else:
```

```

print u" 负类 "
myann.drawponint_add([35,82])
myann.draw2d()

# 下面直接使用默认参数进行训练
traindata2=[[ [9,25,30],-1],[ [5,8,12],-1],[ [15,31,49],-1],[ [35,62,108],
-1],[ [19,40,60],-1],[ [28,65,98],1],[ [20,59,72],1],[ [9,41,38],1],[ [12,60,46],1],[ [2,37,
18],1]]

myann2=nplann.Mplannliner()
myann2.samples_init(traindata2)
myann2.train()
myc=myann2.simulate([35,68,110])
print "[35,68,110]"
if myc==1:
    print u" 正类 "
else:
    print u" 负类 "

```

如图 8-10 所示是程序运行生成的效果图，虚线是分类线，它清晰地将不同类的点分成了上下两部分。星号为测试数据，可以看到，已被正确分类。Rosenblatt 感知器对线性分类效果还是不错的。

**mplannliner** 模块的代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#author: 麦好
#2013-07-25
import numpy as np
import math
import matplotlib.pyplot as plt
class Mplannliner:
    def __init__(self):
        self.b=1
        self.a0=0.1
        self.a=0.0
        self.r=20.0
        self.expect_e=0.05
        self.traincount=100
        self.testpoint=[]
    def testpoint_init(self):
        self.testpoint=[]
    def e_init(self,mye):
        self.expect_e=mye
    def samples_init(self,mysamples):
        my_x=[]
        my_d=[]
        my_w=[self.b]
        myexamp=mysamples
        for mysmpl in myexamp:

```

ANN-LINER[red:+green:-]  
[myhaspl@qq.com]http://blog.csdn.net/u010255642

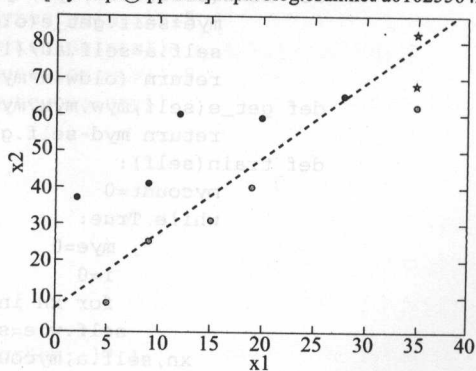


图 8-10 线性分类效果

```

        tempsmp=[1]+mysmp[0]
        my_x.append(tempsmp)
        my_d.append(mysmp[1])
        for i in range(len(my_x[0])-1):
            my_w.append(0.0)
            self.x = np.array(my_x)
            self.d = np.array(my_d)
            self.w = np.array(my_w)

    def a_init(self,mya):
        self.a0=mya

    def r_init(self,myr):
        self.r=myr

    def maxtry_init(self,maxc):
        self.traincount=maxc

    def sgn(self,v):
        if v>0:
            return 1
        else:
            return -1

    def get_v(self,myw,myx):
        return self.sgn(np.dot(myw.T,myx))

    def neww(self,oldw,myd,myx,a,mycount):
        mye=self.get_e(oldw,myx,myd)
        self.a=self.a0/(1+mycount/float(self.r))
        return (oldw+a*mye*myx,mye)

    def get_e(self,myw,myx,myd):
        return myd-self.get_v(myw,myx)

    def train(self):
        mycount=0
        while True:
            mye=0
            i=0
            for xn in self.x:
                self.w,e=self.neww(self.w,self.d[i],\
                    xn,self.a,mycount)
                i+=1
                mye+=pow(e,2)
            mye=math.sqrt(mye)
            mycount+=1
            print u" 第%d次调整中...误差: %f"%(mycount,mye)
            if abs(mye)<self.expect_e or mycount>self.traincount:
                if mycount>self.traincount:
                    print " 已经达到最大训练次数:%d"%mycount
                    break

    def simulate(self,testdata):
        if self.get_v(self.w,np.array([1]+testdata))>0:
            return 1
        else:
            return -1

    def drawponint_add(self,point):
        self.testpoint.append(point)

    def draw2d(self):

```

```

temp_x=[]
temp_y=[]
i=0
for mysamp in self.x:
    temp_x.append(mysamp[1])
    temp_y.append(mysamp[2])
    if self.d[i] > 0:
        plt.plot(mysamp[1],mysamp[2],"or")
    else:
        plt.plot(mysamp[1],mysamp[2],"og")
    i+=1
mytestpointx=[]
mytestpointy=[]
for addpoint in self.testpoint:
    if self.simulate(addpoint)=="+":
        plt.plot(addpoint[0],addpoint[1], '*r')
    else:
        plt.plot(addpoint[0],addpoint[1], '*g')

mytestpointx.append(addpoint[0])
mytestpointy.append(addpoint[1])

x_max=max(max(temp_x),max(mytestpointx))+5
x_min=min(min(temp_x),min(mytestpointx))
y_max=max(max(temp_y),max(mytestpointy))+5
y_min=min(min(temp_y),min(mytestpointy))
if x_min > 0:
    x_min=0
if y_min > 0:
    y_min=0
plt.xlabel(u"x1")
plt.xlim(x_min, x_max)
plt.ylabel(u"x2")
plt.ylim(y_min, y_max)
plt.title("ANN-LINER[red:+ green:-]")
lp_x1 = [x_min, x_max]
lp_x2 = []
myb=self.w[0]
myw1=self.w[1]
myw2=self.w[2]
myy=(-myb-myw1*lp_x1[0])/float(myw2)
lp_x2.append(myy)
myy=(-myb-myw1*lp_x1[1])/float(myw2)
lp_x2.append(myy)
plt.plot(lp_x1, lp_x2, 'b--')
plt.show()

```

### 8.1.3 反向传播与多层感知器

#### 1. 反向传播算法

反向传播算法对梯度下降法进行了改进，主要由两个环节（激励传播、权重更新）反复循环迭代，直到网络对输入的响应达到预定的目标范围为止。它可用来学习多层网络的权

值,通过搜索一个巨大的假设空间(这个空间由网络中所有单元的所有可能权值定义)得到误差曲面。在多层神经网络中,误差曲面存在全局最小值,但也可能存在多个局部极小值,梯度下降法不能保证收敛到全局极小值,但反向传播算法较好地解决了这个问题,在实践中有出色的效果。反向传播算法有以下特点:

1) 网络中的每个神经元模型包括一个非线性激活函数,非线性是光滑的(即处处可微)也是必要的,否则网络的输入输出关系会被归结为单层感知器。

2) 网络包括一层或多层神经元的隐层,它不负责网络的输入输出。这些隐层神经元逐步从输入向量中提取有用特征,使网络学习复杂的任务。

3) 网络的连接强度由网络突触决定。

反向传播算法的主要过程如下:

1) 激励传播过程。在神经元  $j$  的激活函数输入处应用诱导局部域  $v_j(n)$ , 定义如下:

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$$

这个过程完成逐层从输入层传播至输出层的任务。

2) 权重更新过程。对每层的神经元的权值进行修正,公式如下:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

其中,  $\Delta w_{ji}(n)$  是指神经元  $i$  连接到神经元  $j$  的突触  $ji$  的权值的校正值,  $\eta$  是学习率,  $\delta_j(n)$  是局部梯度。

反向传播算法的过程如下:

从建立一个具有期望数量的隐藏单元和输出单元的网络开始,初始化所有的网络的权值为较小的随机数(0 ~ 1 的实数);然后,给定一个固定的网络结构;最后,算法的主循环对训练样例进行反复迭代,对于每一个训练样例,它会应用目前的网络到这个样例中,并计算出对这个样例网络输出的误差,更新网络中所有的权值,然后对这样的梯度下降步骤进行迭代,直到网络的性能达到可接受的精度为止。

但该算法存在局限性,仍然无法摆脱陷入局部最小误差的地步。因此,要在此基础上增加冲量(动量)项。通过修改权值更新法则,使第  $n$  次迭代时权值的更新受到第  $n-1$  次迭代的影响,即本次迭代的更新值的计算,有部分参数来自于上次迭代的更新值,这样,冲量有时会滚过误差曲面的局部极小值,并在梯度不变的区域内逐渐增大搜索步长,加快收敛,从而减少训练次数。

## 2. 多层感知器网络

多层感知器利用非线性神经元作为中间隐藏层神经元,输出端可以直接使用非线性神经元,也可以使用线性神经元。如图 8-11 所示是一个拥有两个隐藏层的多层感知器网络。

在图 8-11 中,网络由输入层、隐藏层及输出层构成,它能够表示种类繁多的非线性曲面,多层网络能在隐藏层自动发现并被有效表示。它允许学习器创造出设计者没有明确引入的特征,网络中使用的单元层越多,可以创造出的特征越复杂。

多层感知器的激活函数及局域梯度如下。



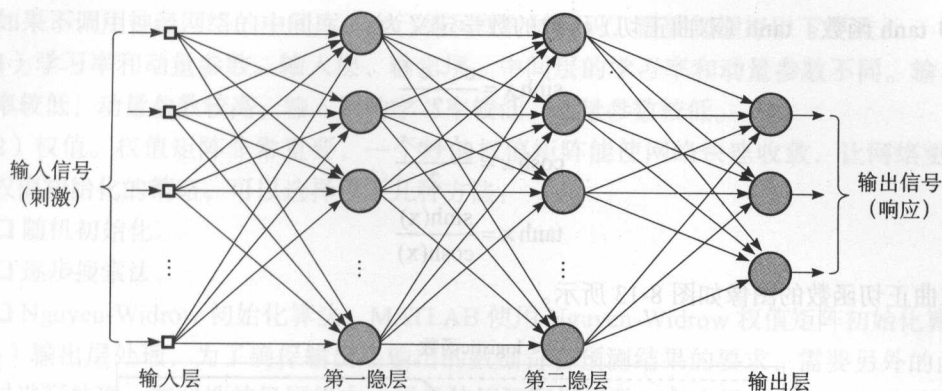


图 8-11 多层感知器网络

1) logistic 函数。logistic 函数的定义为:

$$P(t) = \frac{1}{1 + e^{-t}}$$

它又称 sigmoid 曲线 (S 形曲线), 如图 8-12 所示是它的图像, 很像字母 S。

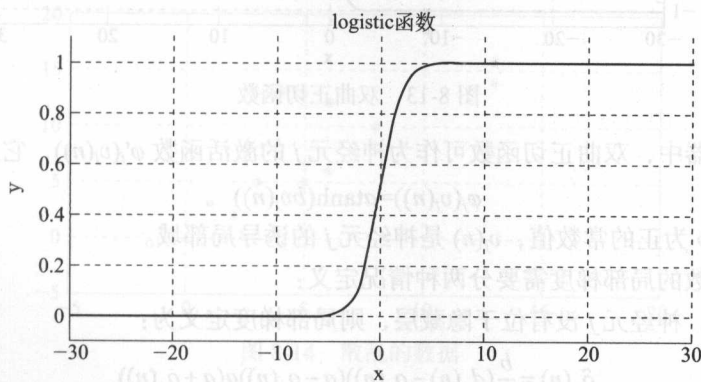


图 8-12 sigmoid 曲线 (附彩图)

在多层感知器中, logistic 函数可作为神经元  $j$  的激活函数  $\varphi'_j(v_j(n))$ , 它定义为:

$$\varphi'_j(v_j(n)) = \frac{1}{1 + \exp(-av_j(n))}, a > 0$$

其中,  $v_j(n)$  是神经元  $j$  的诱导局部域。

logistic 函数的局部梯度需要分两种情况定义:

第一种情况, 神经  $j$  没有位于隐藏层, 则局部梯度定义为:

$$\delta_j(n) = a(d_j(n) - o_j(n))o_j(n)(1 - o_j(n))$$

第二种情况, 神经  $j$  位于隐藏层, 则局部梯度定义为:

$$\delta_j(n) = ay_j(n)(1 - y_j(n)) \sum_k \delta_k(n)w_{kj}(n)$$

2) tanh 函数。tanh (双曲正切) 函数的数学定义为:

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\tanh x = \frac{\sinh(x)}{\cosh(x)}$$

双曲正切函数的图像如图 8-13 所示。

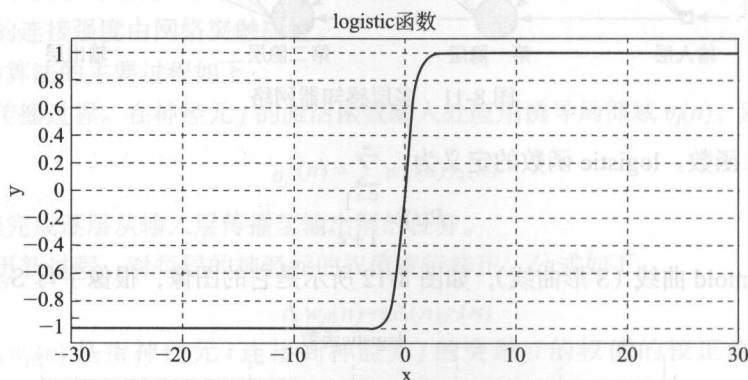


图 8-13 双曲正切函数

在多层感知器中, 双曲正切函数可作为神经元  $j$  的激活函数  $\phi'_j(v_j(n))$ , 它定义为:

$$\phi_j(v_j(n)) = a \tanh(bv_j(n))$$

其中,  $a$  和  $b$  为正的常数值,  $v_j(n)$  是神经元  $j$  的诱导局部域。

双曲正切函数的局部梯度需要分两种情况定义:

第一种情况, 神经元  $j$  没有位于隐藏层, 则局部梯度定义为:

$$\delta_j(n) = \frac{b}{a} (d_j(n) - o_j(n)) (a - o_j(n)) a (a + o_j(n))$$

第二种情况, 神经  $j$  位于隐藏层, 则局部梯度定义为:

$$\delta_j(n) = \frac{b}{a} (d - y_j(n)) (a + y_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

前面说过, 多层感知器利用非线性神经元作为中间隐藏层神经元, 非线性神经元使用双曲正切函数或 logistic 函数作为激活函数, 使用非线性神经元或使用线性神经元组成输出层。多层感知器的训练需要很多样本, 其中每个样本的学习过程分为前向计算和反向计算。

前向计算根据权值矩阵由前向后一层层神经元地推进, 每次推进根据权值计算每层的输出值; 反向计算根据输出结果与目标输出的误差来由后向前调整神经网络的权值, 引入冲量作为神经网络权值调整的依据之一, 冲量 (动量) 参数既可以调整学习速度, 还能体现时间延迟。

整个算法过程也是反向传播算法的过程, 通过使用梯度下降方法搜索可能假设的空间, 迭代减小网络的误差以拟合训练数据。

如果不调用神经网络的中间库，自己编写所有的代码，需要把握好以下要点：

1) 学习率和动量参数。输入层、输出层、中间层的学习率和动量参数不同。输出层的学习率较低，动量参数较高；输入层的学习率较低，动量参数较低。

2) 权值。权值矩阵非常重要，一个好的权值矩阵能使网络快速收敛，让网络更稳定。关于权值初始化的策略，可以选择以下几种方法：

□ 随机初始化。

□ 逐步搜索法。

□ Nguyen-Widrow 初始化算法，MATLAB 使用 Nguyen-Widrow 权值矩阵初始化算法。

3) 输出层处理，为了确保输出层输出的数据符合预测结果的要求，需要另外的函数对输出层进行处理。在线性神经网络中，通常使用硬限幅函数，在多层感知器这种非线性神经网络中既可以使用硬限幅，也可以使用其他函数，具体使用什么函数要看训练效果如何。

4) 数据预处理。输入数据五花八门，在训练之前对数据进行预处理是非常必要的。通过预处理权值矩阵使进入神经网络的数值不会过大或过小，以保证通过中间层的非线性神经元时，输出不逼近其极限。

先来看图 8-14，上面显示的数据散乱且不均匀。

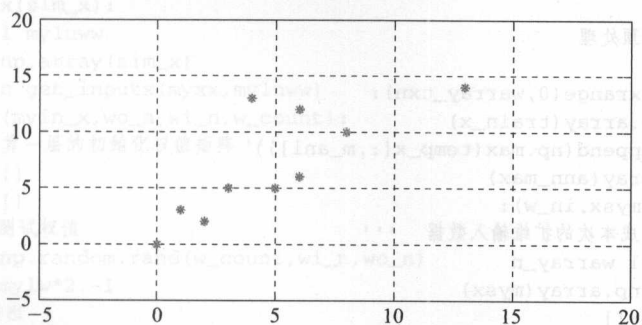


图 8-14 散乱的数据

数据预处理的目标是：将图 8-14 中的数据转化为如图 8-15 所示的形式，数据均匀地分布在坐标系中，4 个象限均有数据存在。

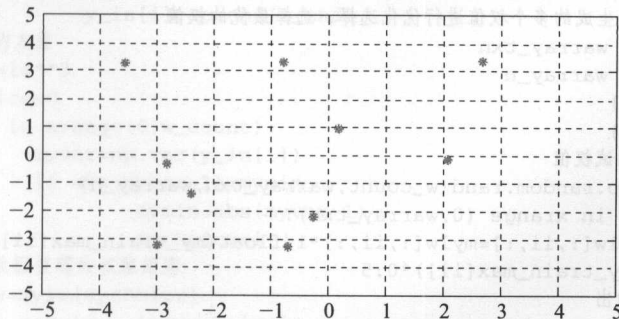


图 8-15 均匀分布的数据

前面讲述了多层感知器的理论基础，下面用 Python 实现一个多层感知器，理论联系实际，这样才能更好地理解反向传播算法和多层感知器。这里要实现的是使用感知器将一组数据进行非线性分类，具体要求为：对下面这组输入数据和输出目标进行训练，对未知数据进行仿真测试。

```
x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
d = [[1,0],[0,1],[1,0],[0,1],[1,0],[0,1]]
```

为了简化实现过程，使用原始的纯随机生成方法产生多层感知器网络的权值矩阵。权值初始值既要保证权值矩阵实现输入项在网络中均匀分布，又要保证权值矩阵本身的均匀分布。为达到这个目的，随机生成若干个权值矩阵，然后从中选择最优化的权值矩阵。最优化的标准为：

1) 对于输入层的权值设计，要尽量使输入数据的方差接近 1。此外，对于相差较大的样本，将它们处理为分布不太接近饱和的可供训练的输入数据。

2) 权值矩阵的均值要尽可能小，其方差尽可能与神经元的突触连接数成反比。

同时，要考虑到数据的预处理，需要在输入层前设置一个预处理权值矩阵，所有的输入经过预处理权值矩阵处理后进入多层感知器的输入层。代码如下：

```
# 对输入数据进行预处理
ann_max=[]
for m_ani in xrange(0,warray_txn):
    temp_x=np.array(train_x)
    ann_max.append(np.max(temp_x[:,m_ani]))
ann_max=np.array(ann_max)
def getnowsx(mysx,in_w):
    '''生成本次的扩维输入数据'''
    global warray_n
    mysx=np.array(mysx)
    x_end=[]
    for i in xrange(0,warray_n):
        x_end.append(np.dot(mysx,in_w[:,i]))
    return x_end

def get_inlw(my_train_max,w_count,myin_x):
    '''计算对输入数据预处理的权值'''
    # 对随机生成的多个权值进行优化选择，选择最优的权值
    global warray_txn
    global warray_n
    mylw=[]
    y_in=[]
    # 生成测试权值
    mylw=np.random.rand(w_count,warray_txn,warray_n)
    for ii in xrange(0,warray_txn):
        mylw[:,ii,:]=mylw[:,ii,:]*1/float(my_train_max[ii])-1/float(my_train_max[ii])*0.5
    # 计算输出
    for i in xrange(0,w_count):
        y_in.append([])
```

```

        for xj in xrange(0,len(myin_x)):
            y_in[i].append(getnowsx(myin_x[xj],mylw[i]))

# 计算均方差
mymin=10**5
mychoice=0
for i in xrange(0,w_count):
    myvar=np.var(y_in[i])
    if abs(myvar-1)<mymin:
        mymin=abs(myvar-1)
        mychoice=i

# 返回数据整理的权值矩阵
return mylw[mychoice]

mylnnw=get_inlw(ann_max,300,train_x)

def get_inputx(mytrain_x,myin_w):
    ''' 将训练数据通过输入权数, 扩维后形成输入数据 '''
    end_trainx=[]
    for i in xrange(0,len(mytrain_x)):
        end_trainx.append(getnowsx(mytrain_x[i],myin_w))
    return end_trainx
x=get_inputx(train_x,mylnnw)
def get_siminx(sim_x):
    global mylnnw
    myxx=np.array(sim_x)
    return get_inputx(myxx,mylnnw)
def getlevelw(myin_x,wo_n,wi_n,w_count):
    ''' 计算一层的初始化权值矩阵 '''
    mylw=[]
    y_in=[]
    # 生成测试权值
    mylw=np.random.rand(w_count,wi_n,wo_n)
    mylw=mylw*2.-1
    # 计算输出
    for i in xrange(0,w_count):
        y_in.append([])
        for xj in xrange(0,len(myin_x)):
            x_end=[]
            for myii in xrange(0,wo_n):
                x_end.append(np.dot(myin_x[xj],mylw[i,:,myii]))
            y_in[i].append(x_end)

# 计算均方差
mymin=10**3
mychoice=0
for i in xrange(0,w_count):
    myvar=np.var(y_in[i])
    if abs(myvar-1)<mymin:
        mymin=abs(myvar-1)
        mychoice=i

# 返回数据整理的权值矩阵
csmylw=mylw[mychoice]
return csmylw,y_in[mychoice]

ann_w=[]

```



```

def init_annw():
    global x
    global hidelevel_count
    global warray_n
    global d
    global ann_w
    ann_w=[]
    lwyii=np.array(x)
    for myn in xrange(0,hidelevel_count):
        # 层数
        ann_w.append([])
        if myn==hidelevel_count-1:
            for iii in xrange(0,warray_n):
                ann_w[myn].append([])
                for jjj in xrange(0,warray_n):
                    ann_w[myn][iii].append(0.0)
        elif myn==hidelevel_count-2:
            templw,lwyii=getlevelw(lwyii,len(d[0]),warray_n,200)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])
                for xjj in xrange(0,len(d[0])):
                    ann_w[myn][xii].append(templw[xii,xjj])
                for xjj in xrange(len(d[0]),warray_n):
                    ann_w[myn][xii].append(0.0)
        else:
            templw,lwyii=getlevelw(lwyii,warray_n,warray_n,200)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])
                for xjj in xrange(0,warray_n):
                    ann_w[myn][xii].append(templw[xii,xjj])
    ann_w=np.array(ann_w)
def generate_lw(trycount):
    global ann_w
    print u"产生权值初始矩阵",
    meanmin=1
    myann_w=ann_w
    alltry=30
    tryc=0
    while tryc<alltry:
        for i_i in range(trycount):
            print ".",
            init_annw()
            if abs(np.mean(np.array(ann_w)))<meanmin:
                meanmin=abs(np.mean(np.array(ann_w)))
                myann_w=ann_w
            tryc+=1
            if abs(np.mean(np.array(myann_w)))<0.008:break
    ann_w=myann_w
    print
    print u"权值矩阵平均:%f"%(np.mean(np.array(ann_w)))
    print u"权值矩阵方差:%f"%(np.var(np.array(ann_w)))
    return ann_w
generate_lw(15)

```

此外,只需要输出一个值,在这里不使用硬限幅函数,而是返回最大值的索引,因此需要编写对输出值进行最后加工的函数(注意,为了由浅入深讲解,从现在直到后面明确声明,误差率的计算以最后此函数的输出结果为标准)。该函数的定义如下:

```
def o_func(myy):
    myresult=[]
    for i in xrange(0,len(myy)):
        mean=np.mean(myy)
        if myy[i]>mean:
            myresult.append(1.0)
        else:
            myresult.append(0.0)
    return np.array(myresult)
```

另外,可选择 `tanh` 函数作为非线性神经元的激活函数,它的输出值在  $[-1,1]$  范围内。下面代码完成激活函数(`ann_atanh` 函数)及其局部梯度(`ann_delta_atanh` 函数)的计算。

```
def ann_atanh(myv):
    atanh_a=1.7159#>0
    atanh_b=2/float(3)#>0
    temp_rs=atanh_a*np.tanh(atanh_b*myv)
    return temp_rs

def ann_delta_atanh(myy,myd,nowlevel,level,n,mydelta,myw):
    anndelta=[]
    atanh_a=1.7159#>0
    atanh_b=2/float(3)#>0
    if nowlevel==level:
        # 输出层
        anndelta=(float(atanh_b)/atanh_a)*(myd-myy)*(atanh_a-myy)*(atanh_a+myy)
    else:
        # 隐藏层
        anndelta=(float(atanh_b)/atanh_a)*(atanh_a-myy)*(atanh_a+myy)
        temp_rs=[]
        for j in xrange(0,n):
            temp_rs.append(sum(myw[j]*mydelta))
        anndelta=anndelta*temp_rs
    return anndelta
```

下面介绍反向传播的核心算法,算法分为前向计算和反向计算两个过程。代码如下:

```
def sample_train(myx,myd,n,sigmoid_func,delta_sigfun):
    ''' 一个样本的前向和后向计算 '''
    global ann_yi
    global ann_delta
    global ann_w
    global ann_wj0
    global ann_y0
    global hidelevel_count
    global alllevel_count
    global learn_r
    global train_a
    global ann_oldw
```

```

level=hidelevel_count
allevel=alllevel_count
# 清空 yi 输出信号数组
hidelevel=hidelevel_count
allevel=alllevel_count
for i in xrange(0,alllevel):
    # 第一维是层数, 从 0 开始
    for j in xrange(0,n):
        # 第二维是神经元
        ann_yi[i][j]=0.0
ann_yi=np.array(ann_yi)
yi=ann_yi
# 清空 delta 矩阵
for i in xrange(0,hidelevel-1):
    for j in xrange(0,n):
        ann_delta[i][j]=0.0
delta=ann_delta
# 保留 w 的拷贝, 以便下一次迭代
ann_oldw=copy.deepcopy(ann_w)
oldw=ann_oldw
# 前向计算
if isdebug:print u"前向计算中..."
# 对输入变量进行预处理
myo=np.array([])
for nowlevel in xrange(0,allevel):
    # 一层层向前计算
    # 计算诱导局域域
    my_y=[]
    myy=yi[nowlevel-1]
    myw=ann_w[nowlevel-1]
    if nowlevel==0:
        # 第一层隐藏层
        my_y=myx
        yi[nowlevel]=my_y
    elif nowlevel==(allevel-1):
        # 输出层
        my_y=o_func(yi[nowlevel-1,:len(myd)])
        yi[nowlevel,:len(myd)]=my_y
    elif nowlevel==(hidelevel-1):
        # 最后一层输出层
        for i in xrange(0,len(myd)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel,:len(myd)]=my_y
    else:
        # 中间隐藏层
        for i in xrange(0,len(myy)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel]=my_y
if isdebug:
    print u"***** 本样本训练的输出矩阵 *****"
    print yi

```

```

print u"*****"
# 计算误差与均方误差
# 因为线性输出层为直接复制，所以取非线性隐藏输出层的结果
myo=yi[hidelevel-1][:len(myd)]
myo_end=yi[alllevel-1][:len(myd)]
mymse=get_e(myd,myo_end)
# 反向计算
# 输入层不需要计算 delta，输出层不需要计算 w
if isdebug:print u"反向计算中..."
# 计算 delta
for nowlevel in xrange(level-1,0,-1):
    if nowlevel==level-1:
        mydelta=delta[nowlevel]
        my_n=len(myd)
    else:
        mydelta=delta[nowlevel+1]
        my_n=n
    myw=ann_w[nowlevel]
    if nowlevel==level-1:
        # 输出层
        mydelta=delta_sigfun(myo,myd,None,None,None,None,None)
        mydelta=mymse*myo
    elif nowlevel==level-2:
        # 输出隐藏层的前一层，传输相当于输出隐藏层的神经元数目的数据
        mydelta=delta_sigfun(yi[nowlevel],myd,nowlevel,level-
1,my_n,mydelta[:len(myd)],myw[:,len(myd)])
    else:
        mydelta=delta_sigfun(yi[nowlevel],myd,nowlevel,level-
1,my_n,mydelta,myw)

    delta[nowlevel][:my_n]=mydelta
# 计算与更新权值 w
for nowlevel in xrange(level-1,0,-1):
    # 每个层的权值不一样
    if nowlevel==level-1:
        # 输出层
        my_n=len(myd)
        mylearn_r=learn_r*0.8
        mytrain_a=train_a*1.6
    elif nowlevel==1:
        # 输入层
        my_n=len(myd)
        mylearn_r=learn_r*0.9
        mytrain_a=train_a*0.8
    else:
        # 其他层
        my_n=n
        mylearn_r=learn_r
        mytrain_a=train_a

    pre_level_myy=yi[nowlevel-1]
    pretrain_myww=oldw[nowlevel-1]
    pretrain_myw=pretrain_myww[:,my_n]

```

```

# 第二个调整参数
temp_i=[]
for i in xrange(0,n):
    temp_i.append([])
    for jj in xrange(0,my_n):
        temp_i[i].append(mylearn_r*delta[nowlevel,jj]*pre_
level_myy[i])
temp_rs2=np.array(temp_i)
temp_rs1=mytrain_a*pretrain_myy
# 总调整参数
temp_change=temp_rs1+temp_rs2
my_ww=ann_w[nowlevel-1]
my_ww[:,my_n]+=temp_change
if isdebug:
    print "=====
    print u"*** 权值矩阵 ***
    print ann_w
    print u"*** 梯度矩阵 ***
    print delta
    print "=====
return mymse

```

还需要训练神经网络，并读取测试数据，验证效果。其中，训练神经网络的代码如下：

```

train()
delta_sigfun=ann_delta_atanh
sigmoid_func=ann_atanh
i=0
for xn in xrange(0,len(x)):
    print u" 样本: %d~%d =>"%(train_x[xn][0],train_x[xn][1])
    print simulate(x[xn],sigmoid_func,delta_sigfun)
    print u"===== 正确目标值 =====
    print d[i]
    i+=1

```

验证神经网络的代码如下：

```

test=np.array(get_siminx([[8,70]]))
print u" 测试值: %f %f"%(8,70)
print simulate(test,sigmoid_func,delta_sigfun)
print u" 正确目标值: [1,0]"
test=np.array(get_siminx([[6.5,272]]))
print u" 测试值: %f %f"%(6.5,272)
print simulate(test,sigmoid_func,delta_sigfun)
print u" 正确目标值: [0,1]"
exitstr=raw_input(u" 按回车键退出 ")

```

运行上述程序，经过 78 次训练，神经网络达到了训练目标，误差率为 0。从以下运行结果来看，效果不错。

```

.....
.....

```



```

----- 开始第 76 次训练 ----- # # # # # 误差为: 0.816497
----- 开始第 77 次训练 ----- # # # # # 误差为: 0.577350
----- 开始第 78 次训练 ----- # # # # # 误差为: 0.000000
训练成功, 正在进行检验
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
训练成功, 误差为: 0.000000
样本: 4~11 =>
仿真计算中
[ 1. 0.]
===== 正确目标值 =====
[1, 0]
样本: 7~340 =>
仿真计算中
[ 0. 1.]
===== 正确目标值 =====
[0, 1]
样本: 10~95 =>
仿真计算中
[ 1. 0.]
===== 正确目标值 =====
[1, 0]
样本: 3~29 =>
仿真计算中
[ 0. 1.]
===== 正确目标值 =====
[0, 1]
样本: 7~43 =>
仿真计算中
[ 1. 0.]
===== 正确目标值 =====
[1, 0]
样本: 5~128 =>
仿真计算中
[ 0. 1.]
===== 正确目标值 =====
[0, 1]
测试值: 8.000000      70.000000
仿真计算中
[ 1. 0.]
正确目标值: [1,0]
测试值: 6.500000      272.000000
仿真计算中
[ 0. 1.]
正确目标值: [0,1]
按回车键退出

```

上述多层感知器的完整源代码见本书资源包中的“多层感知器神经网络源代码.doc”文件或 8-10.py 文件。

前面用例子说明了 Rosenblatt 感知器的局限性，现在在多层感知器上对 Rosenblatt 感知器无法分类的数据进行测试。将“多层感知器神经网络源代码.doc”文件中的代码中的样本数据进行修改，并加上绘制散点图的代码。

```
#x 和 d 样本初始化
train_x = np.array([[1,6],[3,12],[3,9],[3,21],[2,16],[3,15]])
d = np.array([[1,0],[1,0],[0,1],[0,1],[1,0],[0,1]])
.....
for xn in xrange(0,len(x)):
    if simulate(x[xn],sigmoid_func,delta_sigfun)[0] > 0:
        pl.plot(train_x[xn][0],train_x[xn][1],"bo")
    else:
        pl.plot(train_x[xn][0],train_x[xn][1],"b*")
pl.show()
```

运行代码对神经网络进行训练，训练成功后，误差为 0。如图 8-16 所示为训练效果，实心圆圈与星号分别表示两类样本点，多层感知器网络的非线性分类成功完成了 Rosenblatt 感知器无法完成的任务。

现在在上述代码基础上，加上若干随机数据点，将学习率设得较小，进一步观察多层感知器的非线性分类能力。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-11.py
import pylab as pl
import numpy as np
import random
import copy
isdebug=False
#x 和 d 样本初始化
train_x = np.array([[1,6],[3,12],[3,9],[3,21],[2,16],[3,15]])
d = np.array([[1,0],[1,0],[0,1],[0,1],[1,0],[0,1]])
.....
train()
delta_sigfun=ann_delta_atanh
sigmoid_func=ann_atanh
temp_x=np.random.rand(20)*10
temp_y=np.random.rand(20)*20+temp_x
myx=temp_x
myy=temp_y
pl.subplot(111)
x_max=np.max(myx)+5
x_min=np.min(myx)-5
y_max=np.max(myy)+5
```

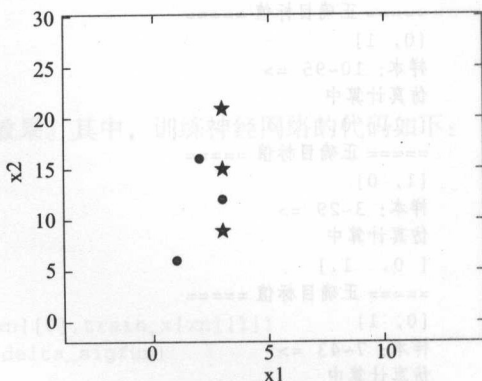


图 8-16 多层感知器网络的非线性分类

```

y_min=np.min(myy)-5
pl.xlabel(u"x1")
pl.xlim(x_min, x_max)
pl.ylabel(u"x2")
pl.ylim(y_min, y_max)
i=0
for mysamp in myx:
    test=get_siminx([[mysamp,myy[i]]])
    if simulate(test,sigmoid_func,delta_sigfun)[0] > 0:
        pl.plot(mysamp,myy[i],"ro")
    else:
        pl.plot(mysamp,myy[i],"g*")
    i+=1
for xn in xrange(0,len(x)):
    if simulate(x[xn],sigmoid_func,delta_sigfun)[0] > 0:
        pl.plot(train_x[xn][0],train_x[xn][1],"bo")
    else:
        pl.plot(train_x[xn][0],train_x[xn][1],"b*")
pl.show()

```

从如图 8-17 所示的分类效果中，能直观感受到多层感知器的非线性分类能力。其中，星号与实心圆圈表示的数据点被分成了两类，它们之间的界限是非线性表示的曲线，而不是线性表示的直线。

在多层感知器学习中，经常需要用到一个概念：误差曲线。误差曲线体现在一个二维坐标系中， $X$ 轴表示训练次数， $Y$ 轴表示每次训练的误差率。理想的误差曲线是随着 $X$ 的增加， $Y$ 值不断平滑减少，这与误差曲面相似，误差曲面以参数为自变量，这些都形象地体现了梯度下降的概念。它们都说明了一个道理：一个设计良好的多层感知器，随着训练次数的增多，模型会越来越精确，误差曲面会朝着最小点向下滑动，而且误差曲线也在下降。

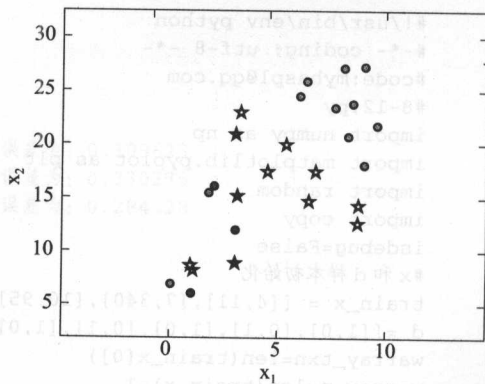


图 8-17 非线性分类

前面为了讲述的需要，“多层感知器神经网络源代码.doc”文件所示代码对误差率的计算以最终输出结果为依据计算，现在在神经网络输出层的后面再增加一层最终输出层，对原输出层的结果进行再次加工后输出。本书在以后涉及的相关代码中，如未特别说明，最终输出结果是指新增的最终输出层的输出结果。在此，将误差率的计算公式定义为：

$$\text{误差率} = \sqrt{\frac{\sum (\text{实际值} - \text{输出值})^2}{\text{样本数目}}}$$

设置好网络的相关参数，将期望误差率设置为 0.3，然后加入绘制误差曲线的代码。在列出代码之前，先介绍一下前面没有具体涉及的问题：中间隐藏层、学习率及动量参数。

一般认为，增加隐藏层数可以降低网络误差，提高精度，但也可使网络复杂化。当然也有学者提出了反对意见，但从实践经验来看，隐藏层的数目过少无法实现对复杂数据的学习，隐藏层的增多增加了网络的训练时间和出现“过拟合”的概率。

设计神经网络至少应考虑3层网络，其中有一层隐藏层，在这里将隐藏层的层数及每层节点数定义为如下形式：

隐藏层的层数 = 每个样本的元素个数  $\times 4 - 2$

每个隐藏层的节点数 = 训练样本数  $- 1$

目前仍有学者在对学习率和动量参数进行研究。什么样的学习率能既加快神经网络的训练时间，同时又能提高训练精度呢？动量参数设置为多少，更适合当前学习率，让误差曲线在下降过程中尽可能地跳出局部最小值的陷阱呢？

如果学习率设得过大，可能造成训练过早停止，错过了误差曲线的全局最小值；而设得过小则会造成训练时间加长，同时容易陷入误差曲线的局部最小值。动量参数也存在类似的问题。笔者认为这些参数需要在实践应用中进行调试和测试，通过对训练效果反复对比，才能得到最适合的参数。

根据以上设计思路，用Python在“多层感知器神经网络源代码.doc”文件代码的基础上实现。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-12.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy
isdebug=False
#x和d样本初始化
train_x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
d = [[1,0],[0,1],[1,0],[0,1],[1,0],[0,1]]
warray_txn=len(train_x[0])
warray_n=len(train_x)-1
#基本参数初始化
oldmse=10**100
fh=1
maxtrycount=500
mycount=0.0
if maxtrycount>=20:
    r=maxtrycount/10
else:
    r=maxtrycount/2
#sigmoid函数
ann_sigfun=None
ann_delta_sigfun=None
#总层数初始化，比非线性导数多一层线性层
alllevel_count=warray_txn*4
#非线性层数初始化
```

```

hidelevel_count=alllevel_count-1
# 学习率参数
learn_r0=0.02
learn_r0*=2.5
learn_r=learn_r0
# 动量参数
train_a0=learn_r0*1.2
train_a0*=0.0015
train_a=train_a0
# 误差率
expect_e=0.3
.....
x_max=len(err)
x_min=1
y_max=max(err)+0.2
y_min=0.
plt.xlabel(u"traincount")
plt.xlim(x_min, x_max)
plt.ylabel(u"mse")
plt.ylim(y_min, y_max)
lp_x1 = xrange(1,len(err)+1)
lp_x2 = err
plt.plot(lp_x1,lp_x2,'g-')
plt.show()

```

运行代码，执行结果如下：

```

.....
----- 开始第 137 次训练 ----- # # # # # 误差为: 0.309623
----- 开始第 138 次训练 ----- # # # # # 误差为: 0.330235
----- 开始第 139 次训练 ----- # # # # # 误差为: 0.284128
训练成功, 正在进行检验
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
训练成功, 输出误差为: 0.000000
样本: 4~11 =>
仿真计算中
[ 1.  0.]
===== 正确目标值 =====
[1, 0]
样本: 7~340 =>
仿真计算中
[ 0.  1.]
===== 正确目标值 =====
[0, 1]
样本: 10~95 =>
仿真计算中
[ 1.  0.]

```



==== 正确目标值 =====

[1, 0]

样本: 3~29 =>

仿真计算中

[ 0. 1.]

==== 正确目标值 =====

[0, 1]

样本: 7~43 =>

仿真计算中

[ 1. 0.]

==== 正确目标值 =====

[1, 0]

样本: 5~128 =>

仿真计算中

[ 0. 1.]

==== 正确目标值 =====

[0, 1]

测试值: 8.000000~70.000000

仿真计算中

[ 1. 0.]

正确目标值: [1,0]

测试值: 6.500000~272.000000

仿真计算中

[ 0. 1.]

正确目标值: [0,1]

从上述代码执行结果中不难看出, 网络在 139 次训练后, 达到了训练误差的期望值 0.3, 对测试数据和样本数据的验证均成功。再来看看如图 8-18 所示的误差曲线, 总体呈现下滑趋势。

图 8-18 中的曲线并不平滑, 如果将学习率设得更小, 曲线将平滑很多, 训练次数也将增多。现在修改代码, 将学习率减少, 重新绘制误差曲线图。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-13.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy
isdebug=False
#x 和 d 样本初始化
train_x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
d = [[1,0],[0,1],[1,0],[0,1],[1,0],[0,1]]
warray_tnx=len(train_x[0])
warray_n=len(train_x)-1
# 基本参数初始化
```

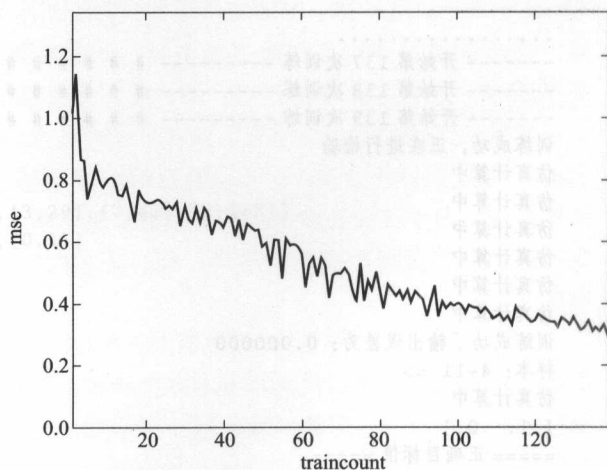


图 8-18 误差曲线 (附彩图)

```

oldmse=10**100
fh=1
maxtrycount=2000
mycount=0.0
if maxtrycount>=20:
    r=maxtrycount/10
else:
    r=maxtrycount/2
#sigmoid 函数
ann_sigfun=None
ann_delta_sigfun=None
# 总层数初始化, 比非线性导数多一层线性层
alllevel_count=warray_txn*4
# 非线性层数初始化
hidelevel_count=alllevel_count-1
# 学习率参数
learn_r0=0.005
learn_r0*=2.5
learn_r=learn_r0
.....
.....

```

程序执行结果如下:

```

----- 开始第 354 次训练 ----- # # # # # 误差为: 0.344559
----- 开始第 355 次训练 ----- # # # # # 误差为: 0.347000
----- 开始第 356 次训练 ----- # # # # # 误差为: 0.338863
----- 开始第 357 次训练 ----- # # # # # 误差为: 0.342834
----- 开始第 358 次训练 ----- # # # # # 误差为: 0.353655
----- 开始第 359 次训练 ----- # # # # # 误差为: 0.280097

```

训练成功, 正在进行检验

```

仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中
仿真计算中

```

训练成功, 输出误差为: 0.000000

样本: 4~11 =>

仿真计算中

[ 1. 0.]

==== 正确目标值 =====

[1, 0]

样本: 7~340 =>

仿真计算中

[ 0. 1.]

==== 正确目标值 =====

[0, 1]

样本: 10~5 =>

仿真计算中

[ 1. 0.]

```
==== 正确目标值 ====
[1, 0]
样本: 3==29 =>
仿真计算中
[ 0.  1.]
==== 正确目标值 ====
[0, 1]
样本: 7~43 =>
仿真计算中
[ 1.  0.]
==== 正确目标值 ====
[1, 0]
样本: 5~128 =>
仿真计算中
[ 0.  1.]
==== 正确目标值 ====
[0, 1]
测试值: 8.000000~70.000000
仿真计算中
[ 1.  0.]
正确目标值: [1,0]
测试值: 6.500000~272.000000
仿真计算中
[ 0.  1.]
正确目标值: [0,1]
```

从执行结果可看出，如果将学习率设得更小，训练次数确实增多了，由 100 多次变成了 300 多次，相比上次的训练，每次训练的误差率降低幅度变小，如图 8-19 所示为误差曲线。

相比图 8-18 的误差曲线，图 8-19 所示的误差曲线下下降趋势更加明显，平滑很多，上下摆动的幅度也减小了不少。

### 8.1.4 Python 神经网络库

前面几节设计和完善了线性和非线性神经网络，并通过 Python 实现了神经网络，逐步阐述了怎样设计一个神经网络，如何改进神经网络，如何应用神经网络完成分类等。后面的章节还将对这些程序代码进行改进，介绍如何应用神经网络进行数据拟合。

“不用重复造轮子”，实践中除非条件限制（比如：受限嵌入式环境等应用）必须编写所有的神经网络代码，否则可以直接调用神经网络相关的库。

目前 Python 关于神经网络的库较多，这里选择纯 Python 库实现的神经网络库 Neurolab，在第 2 章中已经介绍过它的安装与配置方法，在此不再介绍，下面直接讲述如何应用它。首

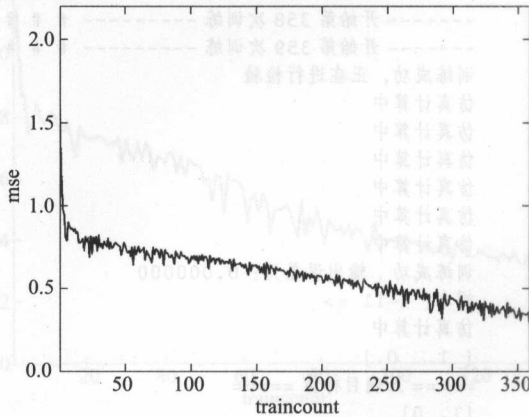


图 8-19 误差曲线

先进入 Python 的控制台，以前几节的数据为例，熟悉一下 Neurolab 的基本使用方法。

```
>>> import neurolab as nl
>>> train_x = [[4,11],[7,340],[10,95],[3,29],[7,43],[5,128]]
>>> input=np.array(train_x)
>>> d =[[1],[0],[1],[0],[1],[0]]
>>> target=np.array(d)
>>> target
array([[1],
       [0],
       [1],
       [0],
       [1],
       [0]])
>>> input
array([[ 4, 11],
       [ 7, 340],
       [10, 95],
       [ 3, 29],
       [ 7, 43],
       [ 5, 128]])
>>> net = nl.net.newff([[3, 10], [11, 400]], [5, 1])
>>> err = net.train(input, target, show=15)
Epoch: 15; Error: 0.326594518922;
Epoch: 30; Error: 0.0242485565317;
The goal of learning is reached
```

训练完毕后，再用数据测试该网络。代码如下：

```
>>> net.sim([[3, 10]])
array([[ 0.99999326]])
>>> net.sim([[9, 80]])
array([[ 0.89054486]])
>>> net.sim([[6.5,272]])
array([[ 0.05707987]])
>>> net.sim([[10,80]])
array([[ 0.9448553]])
>>> net.sim([[5,125]])
array([[ 0.12198103]])
>>> net.sim([[5,100]])
array([[ 0.18880761]])
```

上述代码显示，训练很成功，测试数据也被正确分类。

拥有中间隐藏层的多层感知器在输入与输出之间建立了映射关系，这种映射关系不一定是某种数学模型明确定义的。如果随机生成无规律的输入值和输出值，这些值之前的关系完全未知，那么是不能建立拥有确定数学公式的映射的，但可通过神经网络来建立它们之间的数学模型。代码如下：

```
>>> import numpy as np
>>> import neurolab as nl
>>> input = np.random.uniform(-0.5, 0.5, (10, 2))
```

```
>>> output = np.random.uniform(-0.5, 0.5, (10, 1))
>>> err = net.train(input, output, show=15)
>>> net = nl.net.newff([[[-0.5, 0.5], [-0.5, 0.5]]], [[8, 1]])
>>> err = net.train(input, output, show=15)
Epoch: 15; Error: 0.0676764691815;
Epoch: 30; Error: 0.0131047452439;
The goal of learning is reached
>>> err[len(err)-1]
0.0097660775986454906
```

上述代码中的 `err[len(err)-1]` 表示训练停止后的误差率。可以看到，训练后精度很高，误差率小于 0.01。下面来编写代码绘制误差曲线。

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(err)
[<matplotlib.lines.Line2D object at 0x04C173B0>]
```

如图 8-20 所示是生成的误差曲线，能明显看到误差曲线很平滑。但此处的误差来源数据是训练 15 次采集一次，前几节中的误差曲线是每训练一次采集一次。因此，图 8-20 比前几节的误差曲线更为平滑。

通过多层感知器神经网络，在这组没有联系的随机生成的输入和输出之间“强行”建立了一种映射关系。在测试时，通过未在样本中出现的输入数据，能得到符合这种映射关系的输出。这种映射能力非常重要，是后面几章提到的数据拟合、模式识别等机器学习任务的基础。

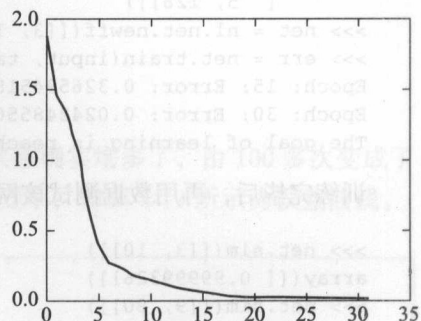


图 8-20 误差曲线

## 8.2 统计算法

统计分析算法在机器学习中占有重要地位，在此，仅介绍几种常用算法。

### 8.2.1 平均值

平均值是统计学中最常用的统计量，用来表明资料中各观测值集中较多的中心位置，用于反映现象总体的一般水平，或分布的集中趋势。有限总体的平均值定义为：

$$\mu = \sum_{i=1}^n x_i / N$$

式中， $\mu$  表示总体平均值， $N$  表示总体所包含的个体数。

平均值的意义在于：不仅可用它来反映一组数据的一般情况，还可以进行不同数据组之间的比较，以看出组与组之间的差别。下面随机生成两组随机数，对它们进行分析。代码如下：

```
>>> y = np.random.uniform(-0.5, 0.5, (10, 1))
>>> x = np.random.uniform(-0.5, 0.5, (10, 1))
>>> x
```



```
array([[ -0.46954884],
       [ -0.39078561],
       [  0.05531789],
       [ -0.06414516],
       [ -0.00511995],
       [ -0.41105398],
       [ -0.23416933],
       [  0.1369419 ],
       [ -0.45076555],
       [  0.1808625 ]])
```

```
>>> y
array([[ 0.38100971],
       [ 0.12510564],
       [ 0.0540655 ],
       [-0.25050503],
       [ 0.13180995],
       [ 0.32953768],
       [-0.35940215],
       [-0.00294618],
       [-0.23359633],
       [-0.49808591]])
>>> np.mean(y)
-0.032300713545130311
>>> np.mean(x)
-0.16524661414915703
```

观察上述代码的执行结果可以看到,  $y$  的平均值为  $-0.032300713545130311$ ,  $x$  的平均值为  $-0.16524661414915703$ ,  $y$  的平均值更趋向于 0, 因此, 我们预言:  $y$  的总体分布要比  $x$  偏右, 更趋向于 0。继续在 Python 控制台中绘制它们在坐标系中的分布, 以验证这个预言。代码如下:

```
>>> z=np.zeros(10)
>>> z
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.])
>>> xx=np.array(zip(z,x));yy=np.array(zip(z,y))
>>> xx
array([[ 0.      , -0.46954884],
       [ 0.      , -0.39078561],
       [ 0.      ,  0.05531789],
       [ 0.      , -0.06414516],
       [ 0.      , -0.00511995],
       [ 0.      , -0.41105398],
       [ 0.      , -0.23416933],
       [ 0.      ,  0.1369419 ],
       [ 0.      , -0.45076555],
       [ 0.      ,  0.1808625 ]])
>>> import matplotlib.pyplot as plt
>>> plt.xlim(-0.5, 0.5)
(-0.5, 0.5)
>>> plt.ylim(-0.01, 0.01)
(-0.01, 0.01)
```

```
>>> plot(yy[:,0],yy[:,1],'or')
[<matplotlib.lines.Line2D object at 0x04EB44D0>]
>>> plot(xx[:,0],xx[:,1],'*b')
[<matplotlib.lines.Line2D object at 0x04CE2810>]
>>>
```

$x$  用星号表示,  $y$  用实心圆圈表示, 绘制图形如图 8-21 所示。从图 8-21 所示的效果图可以看出, 星号表示的数据比实心圆圈表示的数据整体更偏左, 更偏向  $X$  轴的负方向, 这个趋势证实了我们刚才的预言。

## 8.2.2 方差与标准差

### 1. 标准差

标准差是一组数据平均值分散程度的一种度量方式, 其计算公式为:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

其中,  $\mu$  为平均值。

较大的标准差, 代表大部分数值和其平均值之间差异较大; 较小的标准差, 代表这些数值较接近平均值。

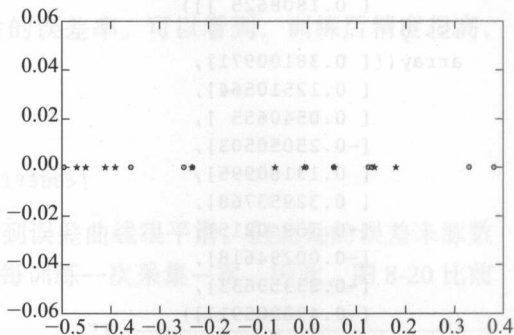


图 8-21 数据点分布

### 2. 方差

标准差的平方就是方差, 其计算公式为:

$$\sigma^2 = \frac{\sum_{i=1}^N (x_i - \mu)^2}{N}$$

其中,  $\mu$  为平均值。

方差与标准差的统计意义差不多, 都是反映数据平均值分散程度。

### 3. Python 实现

下面用 Python 来实现标准差和方差的计算。在 Python 控制台操作, 随机生成一组  $x$  和  $y$  值,  $y$  是均匀分布,  $x$  是高斯分布。代码如下:

```
>>> y = np.random.rand(100)
>>> x = np.random.normal(0.5, 0.00001, 100)
>>> import matplotlib.pyplot as plt
```

观察  $x$  的分布趋势, 如下所示:

```
>>> z=np.zeros(100)
>>> xx=np.array(zip(x,z))
>>> plot(xx[:,0],xx[:,1],'*b')
[<matplotlib.lines.Line2D object at 0x04E06BF0>]
```

$x$  的分布如图 8-22 所示, 数据点在  $X$  轴上分布不均匀, 主要点集中在 0.5 附近 (图 8-22

的基值是  $4.9997\text{e-}1$ ), 少量点散布在两边, 这是预料之中的。

$x$  是高斯分布, 随机生成  $x$  时, 指定了参数为: 平均值 0.5, 最大标准差为 0.000 01。高斯分布就是正态分布, 具有集中性的特点, 即: 正态曲线的高峰位于正中央, 即平均值所在的位置, 如果是一维空间, 则集中在  $X$  轴中平均值附近。

观察  $y$  的分布趋势, 如下所示:

```
>>> z=np.zeros(100)
>>> yy=np.array(zip(y,z))
>>> plot(yy[:,0],yy[:,1],'*b')
[<matplotlib.lines.Line2D object at 0x04E06BF0>]
```

从分布图 8-23 来看,  $y$  均匀分布在  $X$  轴上, 没出现大量数据点聚集的情况。

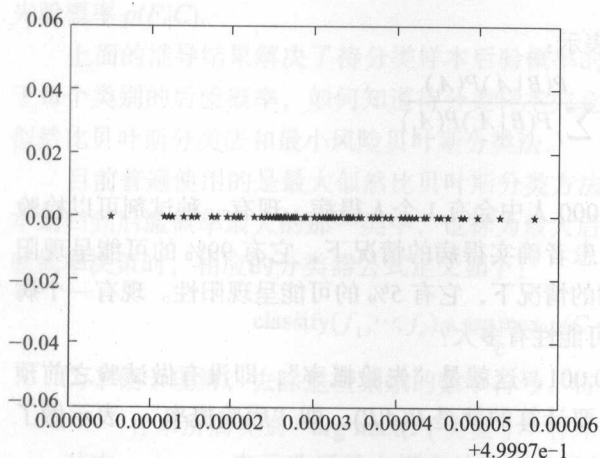


图 8-22 数据点分布

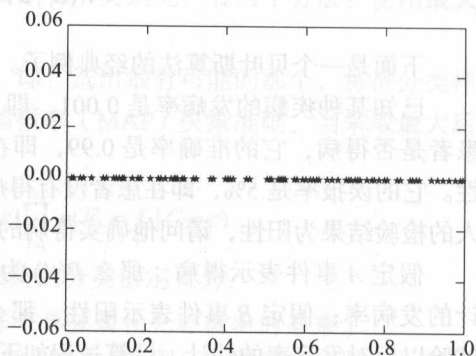


图 8-23 数据点分布 (彩附图)

现在分别求出  $x$  和  $y$  的平均值、方差与标准差。

```
>>> np.mean(x)# 平均值
0.500000063672402462
>>> np.var(x)# 方差
9.791184624177845e-11
>>> np.std(x)# 标准差
9.8950414977289737e-06
>>> np.mean(y)# 平均值
0.52212510115195176
>>> np.std(y)# 标准差
0.28755684683792165
>>> np.var(y)# 方差
0.082688940163367933
```

$y$  的平均值与  $x$  接近, 但其方差和标准差不一样,  $x$  的方差和标准差都比  $y$  小很多, 比  $x$  小很多, 这说明  $x$  相对于  $y$  来说, 有更多的数值集中在平均值 0.5 附近, 这就是方差和标准差的统计意义。

## 8.2.3 贝叶斯算法

### 1. 贝叶斯定理

贝叶斯定理是关于随机事件  $A$  和  $B$  条件概率的一则定理，下式定义了事件  $B$  发生的情况下事件  $A$  发生的条件概率：

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

假设  $\{A_i\}$  是事件集合里的部分集合，其中， $A_1, A_2, \dots, A_n$  是某个过程中若干可能的前提，则  $P(A_i)$  是对各前提条件出现可能性的事先估计，称之为先验概率。如果在这个过程得到了结果  $B$ ，贝叶斯公式定义了  $P(A_i|B)$ ，它是对以  $B$  为前提下  $A_i$  出现概率的估计，则称  $P(A_i|B)$  为后验概率。

对于任意的  $A_i$ ，贝叶斯定理用下式来表示：

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_j P(B|A_j)P(A_j)}$$

下面是一个贝叶斯算法的经典例子。

已知某种疾病的发病率是 0.001，即 1000 人中会有 1 个人得病，现有一种试剂可以检验患者是否得病，它的准确率是 0.99，即在患者确实得病的情况下，它有 99% 的可能呈现阳性。它的误报率是 5%，即在患者没有得病的情况下，它有 5% 的可能呈现阳性。现有一个病人的检验结果为阳性，请问他确实得病的可能性有多大？

假定  $A$  事件表示得病，那么  $P(A)$  为 0.001，这就是“先验概率”，即没有做试验之前预计的发病率；假定  $B$  事件表示阳性，那么要计算的就是  $P(A|B)$ ，即“后验概率”，表示做了试验以后对发病率的估计。计算过程如下：

$$P(A|B) = P(A) \frac{P(B|A)}{P(B|A)P(A) + P(B|\bar{A})P(\bar{A})}$$

$$P(A|B) = 0.01 \times \frac{0.99}{0.99 \times 0.001 + 0.05 \times 0.999} \approx 0.019$$

通过计算，得知  $P(A|B)$  约等于 0.019，即使检验呈现阳性，病人得病的概率也只是不到 2%，也就是说，呈现“假阳性”，阳性结果完全不足以说明病人得病。

贝叶斯算法在机器学习中主要用于分类，其基本原理是：已知样本  $X$ ，首先计算  $P(X|C_i)$ ，得出  $C_i$  类别包含样本  $X$  的先验概率；然后根据贝叶斯定理求后验概率  $P(C_i|X)$ ，得到  $X$  属于  $C_i$  类别的后验概率；最后根据最大后验概率判断所属类别。

### 2. 朴素贝叶斯

贝叶斯算法的基础是概率推理，是在各种条件的存在不确定、仅知其出现概率的情况下，完成推理和决策任务。而朴素贝叶斯算法是基于独立假设的，即假设样本每个特征与其他特征都不相关。

尽管实际上独立假设常常是不准确的，但朴素贝叶斯分类器的若干特性让其在实践中能



够取得令人惊奇的效果, 各类条件特征之间的解耦意味着每个特征的分布都可以独立地被当作一维分布来估计, 减轻了由于维数灾带来的阻碍, 避免了样本规模呈指数增长。在本书的第10章中有关于朴素贝叶斯算法的应用实例。

朴素贝叶斯算法通常在自动分类中使用, 算法的主要过程为: 首先, 计算待分类样本特征的后验概率, 然后使用最大似然比贝叶斯分类法或最小风险贝叶斯分类法完成分类。

当待分类样本拥有若干特征变量  $F_1, F_2, \dots, F_n$  时, 该待分类样本属于类  $C$  的后验概率为:

$$p(C | F_1, \dots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^n p(F_i | C)$$

其中, 证据因子  $Z$  是一个仅依赖  $F_1, \dots, F_n$  的缩放因子, 当特征变量的值已知时是一个常数。

在朴素贝叶斯算法中, 后验概率仅依赖于两个因素: 类先验概率  $p(C)$  和基于独立假设的先验概率  $p(F_i | C)$ 。

上面的推导结果解决了待分类样本后验概率的计算问题, 余下的问题是: 既然得到了属于每个类别的后验概率, 如何知道待分类样本究竟属于哪个类别呢? 有两个方法: 使用最大似然比贝叶斯分类法和最小风险贝叶斯分类法。

目前普遍使用的是最大似然比贝叶斯分类方法, 即: 选出最有可能的那个, 将待分类样本划归到后验概率最大的那一类中, 也称为最大后验概率 (MAP) 决策准则。当采取最大后验概率决策时, 相应的分类器公式定义如下:

$$\text{classify}(f_1, \dots, f_n) = \arg \max_C p(C = c) \prod_{i=1}^n p(F_i = f_i | C = c)$$

为了便于理解, 去除这些繁杂的数学符号, 将最终的分类器定义为:

样本所属类别 =  $\arg \max (P(\text{类型}) \times \text{样本每个单独属性先验概率的累乘})$

其中,  $\arg \max$  表示选择最大概率的类别为最终类别,  $P(\text{类别})$  为类别先验概率, 是指该类别本身的可能性有多少。

朴素贝叶斯虽然是基于独立假设的, 但实践证明这种方法确实很有效。创建了 Viaweb (1998 年, Viaweb 成为最流行的电子商务软件, 被雅虎收购, 改称雅虎商店) 的美国著名程序员 Paul Graham 提出使用朴素贝叶斯识别垃圾邮件的方法, 它通过使用字词等标记与垃圾邮件、非垃圾邮件的关联, 计算一封邮件为垃圾邮件的可能性, 以实现对垃圾邮件的判别。有兴趣的读者可以查阅下面网址:

<http://www.paulgraham.com/spam.html>

一个有趣的例子如下: 假设有一台智能机器, 负责将一堆未知水果分为 3 类: 苹果、桂圆、香蕉。已知, 这一堆水果中, 苹果的数量约占 60%, 桂圆的数量约占 35%, 香蕉的数量约占 5%。

现在需要一位工程师为这台机器设计一个机器学习算法, 这位工程师选择了朴素贝叶斯分类算法, 将算法过程定义为如下形式:

首先, 计算  $P(\text{类别})$ 。

$$P(\text{苹果})=0.6, P(\text{桂圆})=0.35, P(\text{香蕉})=0.05$$



然后,分别准备几个苹果、桂圆、香蕉的样本,分析其重量、颜色、形状,得出它们的先验概率。接着,重头戏开始了,识别未知水果。机器面前摆了一个这样的水果(用人类的“智能”,识别出这是一个香蕉,但机器识别出来不容易,目前人工智能水平远没达到期望的水平),如图 8-24 所示。

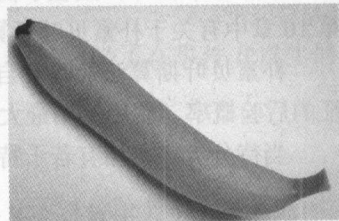


图 8-24 香蕉

机器要做的识别过程是:

1) 提取这个未知样本的特征:重量、颜色、形状。重量测量使用称重仪器,颜色靠色敏元件,形状使用图像识别算法。

2) 从存储器中取出事先计算好的重量、颜色、形状的先验概率,设这个未知水果的重量、颜色、形状特征值为  $a_1$ 、 $a_2$ 、 $a_3$ ,它们各自的先验概率为:

$$P(a_1|\text{苹果})=0.6, P(a_2|\text{苹果})=0.4, P(a_3|\text{苹果})=0.001$$

$$P(a_1|\text{桂圆})=0.001, P(a_2|\text{桂圆})=0.9, P(a_3|\text{桂圆})=0.001$$

$$P(a_1|\text{香蕉})=0.6, P(a_2|\text{香蕉})=0.9, P(a_3|\text{香蕉})=0.95$$

其中,  $a_1$ =重量,  $a_2$ =颜色,  $a_3$ =形状。

3) 计算后验概率。

设这个未知水果为  $x$ , 计算  $x$  属于每个类别的概率。

$$P(\text{苹果}|x)=P(\text{苹果}|a_1, a_2, a_3)$$

$$=P(\text{苹果}) \times (P(a_1|\text{苹果}) \times P(a_2|\text{苹果}) \times P(a_3|\text{苹果}))$$

$$=0.6 \times (0.6 \times 0.4 \times 0.001)$$

$$=0.6 \times 0.00024$$

$$=0.000144$$

$$P(\text{桂圆}|x)=P(\text{桂圆}|a_1, a_2, a_3)$$

$$=P(\text{桂圆}) \times (P(a_1|\text{桂圆}) \times P(a_2|\text{桂圆}) \times P(a_3|\text{桂圆}))$$

$$=0.35 \times (0.001 \times 0.9 \times 0.001)$$

$$=0.000000315$$

$$P(\text{香蕉}|x)=P(\text{香蕉}|a_1, a_2, a_3)$$

$$=P(\text{香蕉}) \times (P(a_1|\text{香蕉}) \times P(a_2|\text{香蕉}) \times P(a_3|\text{香蕉}))$$

$$=0.05 \times (0.6 \times 0.9 \times 0.95)$$

$$=0.02565$$

4) 寻找最大后验概率得到所属类别,假设所属类别为  $C$ 。

$$P(C|x)=\operatorname{argmax}(P(\text{苹果}|x), P(\text{桂圆}|x), P(\text{香蕉}|x))$$

$$=\operatorname{argmax}(0.000144, 0.000000315, 0.02565)$$

3个概率中,0.02565是最大的概率,因此  $C$ =香蕉。

5) 将这个水果分到香蕉堆中。

在本书的 12.3 节将进一步讲解贝叶斯算法。

## 8.3 欧氏距离

### 1. 数学原理

以  $R$  表示实数域。对任意一个正整数  $n$ ，实数的  $n$  元组的全体构成了  $R$  上的一个  $n$  维向量空间，用  $R^n$  来表示，也称之为实数坐标空间。 $R^n$  中的元素写作  $X=(x_1, x_2, \dots, x_n)$ ，这里的  $x_i$  都是实数。欧氏范数定义  $R^n$  上的距离函数（或称度量）：

$$d(x, y) = \|x - y\| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

这个距离函数称为欧几里得度量，也称欧氏距离。欧氏距离通常用于衡量两个点之间的距离，这两个点可以是定义在二维空间的，也可以是定义在三维空间或者  $n$  维空间的。

### 2. 计算实例

假设在二维空间中，已定义两个点：(3,8) 和 (2,5)，其欧氏距离如下：

$$d = \sqrt{(3-2)^2 + (8-5)^2} = 3.1623$$

经过计算，该距离为 3.1623，两点在空间的表示如图 8-25 所示。该距离为直线的长度。

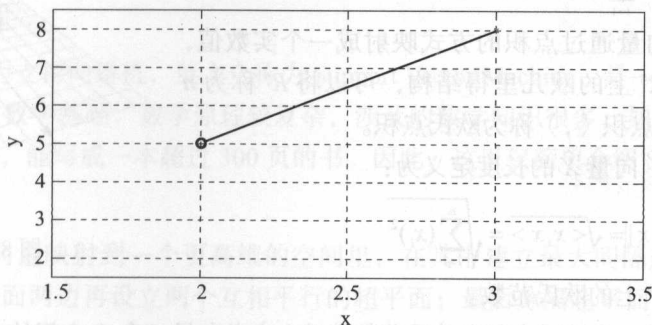


图 8-25 二维空间中两点的距离

假设有两点：(2,5,7) 和 (3,8,2)，它们在三维空间的欧氏距离定义为：

$$d = \sqrt{(3-2)^2 + (5-8)^2 + (7-2)^2} = 5.9161$$

经过计算，该距离为 5.9161，如图 8-26 所示，该距离为直线的长度。

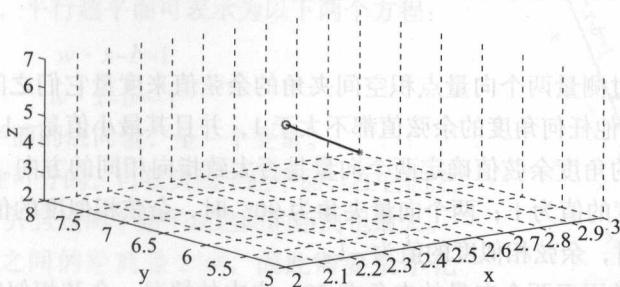


图 8-26 三维空间中两点的距离

$n$  维欧氏空间是一个点集, 它的每个点  $X$  可以表示为  $(x[1], x[2], \dots, x[n])$ , 其中  $x[i]$  ( $i=1, 2, \dots, n$ ) 是实数, 称为  $X$  的第  $i$  个坐标, 两个点  $A=(a[1], a[2], \dots, a[n])$  和  $B=(b[1], b[2], \dots, b[n])$  之间的距离  $d(A, B)$  计算方式如下:

$$d(A, B) = \sqrt{\sum (a[i] - b[i])^2}$$

## 8.4 余弦相似度

### 1. 数学原理

1) 向量。空间中有两个点  $A$  和  $B$ ,  $A \rightarrow B$  就是一个向量, 可以读成从  $A$  到  $B$ 。它既有大小又有方向, 设原点是  $O$ , 给定空间中任意一点  $A$ ,  $OA$  是从  $O$  到  $A$  的向量。如图 8-27 所示就是一个定义在三维空间上的向量  $OA$ 。

2) 点积 (内积)。对任意两个向量  $x, y$ , 点积  $\langle x, y \rangle$  ( $x \cdot y$ ) 定义为:

$$\langle x, y \rangle = \sum_{i=1}^n x_i y_i = x_1 y_1 + x_2 y_2 + \dots + x_n y_n$$

$R^n$  中的两个向量通过点积的方式映射成一个实数值,  $R^n$  及其点积称为  $R^n$  上的欧几里得结构, 可以将  $R^n$  称为  $n$  维欧几里得空间, 点积  $\langle \cdot, \cdot \rangle$  称为欧氏点积。

3) 向量长度。向量  $X$  的长度定义为:

$$\|x\| = \sqrt{\langle x, x \rangle} = \sqrt{\sum_{i=1}^n (x_i)^2}$$

上式又称为  $R^n$  上的欧氏范数。

4) 余弦相似性。在欧氏内积和欧氏范数基础上定义向量  $A$  和  $B$  之间的  $\theta$  余弦相似性如下:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

其中  $\theta$  为  $x$  和  $y$  所夹的内角。

### 2. 算法原理

余弦相似性通过测量两个向量点积空间夹角的余弦值来度量它们之间的相似性。0 度角的余弦值是 1, 而其他任何角度的余弦值都不大于 1, 并且其最小值是 -1。

两个向量之间的角度余弦值确定两个向量是否大致指向相同的方向。两个向量有相同的指向时, 余弦相似度的值为 1; 两个向量夹角为  $90^\circ$  时, 余弦相似度的值为 0; 两个向量指向完全相反的方向时, 余弦相似度的值为 -1。

余弦相似度通常用于两个向量的夹角在  $90^\circ$  之内的情况, 余弦相似度的值为 0 ~ 1。余

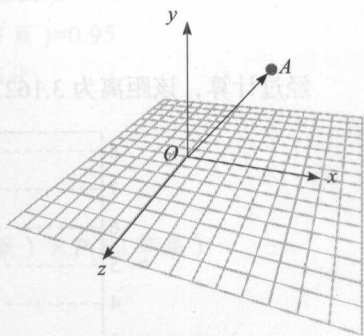


图 8-27 向量

弦相似度可在任何维度的向量比较中,因此在高维空间中的应用非常广泛。

在实际应用中,通常会先提取两个数据的特征,每个数据各形成一个 $n$ 维向量,然后通过计算这两个向量的余弦相似度来判定这两个向量是否是同一类型。例如:文本分类、图像识别等都能使用余弦相似度算法。

设定一个常数 $C$  ( $0 \leq C \leq 1$ ),  $OA$  与  $OB$  之间形成了夹角 $a$ ,余弦相似度如果大于 $C$ ,就认为 $OA$ 与 $OB$ 属于同一类,如图8-28所示。

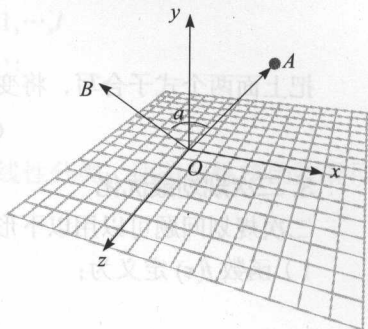


图 8-28 余弦相似度

## 8.5 SVM

1992 ~ 1995 年, Vapnik 等在 SLT 的基础上发展了 SVM 算法。它在解决小样本、非线性及高维模式识别问题中都表现出了许多特有的优势,目前已经成为与神经网络地位齐名的算法,在样本量较小的情况下,其实际运用效果甚至超过了神经网络。

### 8.5.1 数学原理

SVM 中文名为支持向量机,英文全称为 Support Vector Machine,是一种监督式学习的方法。它拥有坚实的数学基础,数学原理较复杂,涉及的数学知识很多,如果将其相关理论以及推导完全展开来,能写成一本超过 300 页的书。因此,这里仅简单介绍 SVM 的基本原理。

#### 1. 算法策略

SVM 首先将向量映射到一个更高维的空间里,在其中建立最大间隔超平面,将数据分开;然后,在超平面两边再设立两个互相平行的超平面;最后分隔超平面,使两个平行超平面的距离最大化。SVM 假定平行超平面间的距离或差距越大,分类器的总误差越小。

#### 2. 超平面

超平面的数学形式可以写作:

$$w \cdot x - b = 0$$

其中 $x$ 是超平面上的点, $w$ 是垂直于超平面的向量。

由图8-29可知,平行超平面可表示为以下两个方程:

$$w \cdot x - b = 1$$

$$w \cdot x - b = -1$$

其中, $w$ 为超平面的法向量,是一个变量。

如果数据是线性可分的,可找到两个超平面,在它们之间没有任何样本点,并且这两个超平面之间的距离也最大。

这两个超平面之间的距离是 $2/|w|$ ,因此需要最小化 $|w|$ ,因为这两个超平面之间没有任何样本点,所以 $x_i$ 还需

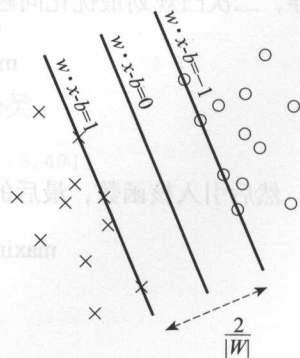


图 8-29 超平面



要满足以下两个条件中的一个：

$$w \cdot x_i - b \geq 1$$

$$w \cdot x_i - b \leq -1$$

把上面两个式子合写，将变成：

$$C_i(w \cdot x_i - b) \geq -1, \quad 1 \leq i \leq n$$

### 3. 二次规划最优化

二次规划问题可以用以下形式来描述。

1) 函数  $f(x)$  定义为：

$$f(x) = (1/2)x^T Qx + c^T x$$

其中， $x^T$  是  $x$  的转置。

2) 函数受到一个或更多如下形式的限制条件：

$$Ax \leq b$$

$$Ex = d$$

如果  $Q$  是半正定矩阵，那么  $f(x)$  是一个凸函数。如果有至少一个向量  $x$  满足约束而且  $f(x)$  在可行域有下界，二次规划问题就有一个全局最小值  $x$ 。如果  $Q$  是正定矩阵，那么全局最小值就是唯一的。如果  $Q=0$ ，二次规划问题就变成线性规划问题。

再来看 SVM 算法，该算法提出了两个要求：第一，超平面之间没有任何样本点；第二，超平面之间的距离最大。这样就形成了二次规划最优化问题。

$$\text{minimize: } W(a) = \frac{\|w\|^2}{2} + C \sum_i \xi_i$$

$$\text{subject to } y_i(w^T x_i + b) \geq 1 - \xi_i, \forall i$$

$$\xi_i \geq 0, \forall i$$

这也是一个二次凸规划问题。

根据优化理论，一个点  $x$  成为全局最小值的必要条件是满足 Karush-Kuhn-Tucker (KKT) 条件，当  $f(x)$  是凸函数时，KKT 条件也是充分条件。因此通过拉格朗日变换以及 KKT 定理推导，二次凸规划最优化问题转变为：

$$\text{maximize: } W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i x_j$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \sum_i \alpha_i y_i = 0$$

然后引入核函数，最后的目标函数为：

$$\text{maximize: } W(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{subject to: } \sum_{i=1}^n \alpha_i y_i = 0, 0 \leq \alpha_i \leq C, \forall i$$

改写为矩阵相乘的格式，得到：



$$\text{minimize: } f(\alpha) = \frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

$$\text{subject to } y^T \alpha = 0 \quad 0 \leq \alpha_i \leq C, i=1, \dots, l$$

最终的目标变成了：通过训练样本寻找  $\alpha$ ，使得下式最小：

$$\frac{1}{2} \alpha^T Q \alpha - e^T \alpha$$

核函数可以是线性的或非线性的，线性核函数仅能完成线性分类，而非线性核函数既可以完成线性分类，也可以完成非线性分类。

### 8.5.2 SMO 算法

SMO（最小贯序列方法）是改进的 SVM 训练算法，同其他 SVM 训练算法一样，SMO 会将一个大的 QP 问题分解为一系列小的 QP 问题。与其他算法不一样的是，SMO 处理的是规模最小的 QP 问题，因此能够快速解决并获得解析解，而且能够有效地改善空间和时间复杂度。

SMO 基本原理是：每次仅选择两个元素共同优化，在其他参数固定的前提下，找到这两个参数的最优值，并更新相应的  $\alpha$  向量，而这两个点乘子的优化可以获得解析解。

### 8.5.3 算法应用

SVM 算法的数学原理较难懂，推导过程复杂，掌握 SVM 算法的关键在于动手实践。通过编写代码，运用 SVM 算法解决机器学习问题才能真正理解它。mlpy 库提供了 SVM 算法的相关函数，本节将以 mlpy 库为例进行阐述，请按第 2 章的指导安装 mlpy 库。

#### 1. 核函数

mlpy 的 SVM 算法库提供以下核函数。

线性核函数通常表现为直线方程，函数名为 linear。

非线性核函数如下：

- 1) 多项式函数，函数名为 poly。
- 2) 径向基函数，函数名为 rbf。
- 3) sigmoid 函数，函数名为 sigmoid。

#### 2. 线性分类

下面使用线性核作为 SVM 的核函数，对下面的数据进行分类。

```
x = [[1,8],[3,20],[1,15],[3,35],[5,35],[4,40],[7,80],[6,49]]
y = [1,1,0,0,1,0,0,1]
```

创建 SVM 类的实例，并使用 learn 方法训练 SVM。

```
x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm()
svm.learn(x, y)
```

使用 `pred` 方法对未知数据进行分类。代码如下：

```
ty=svm.pred(tlp_x[ii])
```

完整的 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#email:myhaspl@qq.com
#author: 麦好
#8-14.py
import numpy as np
import matplotlib.pyplot as plt
import mlpy
print 'loading ...'

x = [[1,8],[3,20],[1,15],[3,35],[5,35],[4,40],[7,80],[6,49]]
y=[1,1,0,0,1,0,0,1]
showpoint=['ro','r*']
tshowpoint=['bo','b*']
x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm()
svm.learn(x, y)
lp_x1 = x[:,0]
lp_x2 = x[:,1]
xmin, xmax = np.min(lp_x1)-1, np.max(lp_x1)+1
ymin, ymax = np.min(lp_x2)-1, np.max(lp_x2)+1
plt.subplot(111)
plt.xlabel(u"x")
plt.xlim(xmin, xmax)
plt.ylabel(u"y")
plt.ylim(ymin, ymax)
# 显示样本点
for ii in xrange(0,len(x)):
    ty=svm.pred(x[ii])
    if ty>0:
        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])
    else:
        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])

# 未知样本分类
tlp_x1=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x2=np.random.rand(50)*(ymax-ymin)+xmin
tlp_x=np.array(zip(tlp_x1,tlp_x2))
for ii in xrange(0,len(tlp_x)):
    ty=svm.pred(tlp_x[ii])
    if ty>0:
        plt.plot(tlp_x1[ii],tlp_x2[ii], tshowpoint[int(ty)])
    else:
        plt.plot(tlp_x1[ii],tlp_x2[ii], tshowpoint[int(ty)])
plt.show()
```

上述代码中，随机生成了 50 个数据作为待分类未知样本。如图 8-30 所示是程序生成的

分类散点图，实心圆圈和星号分别代表两类数据。运行程序后，将出现图 8-30 的彩图，其中红色的是样本数据，蓝色的是测试数据。从图上能直观看到，分类很成功。

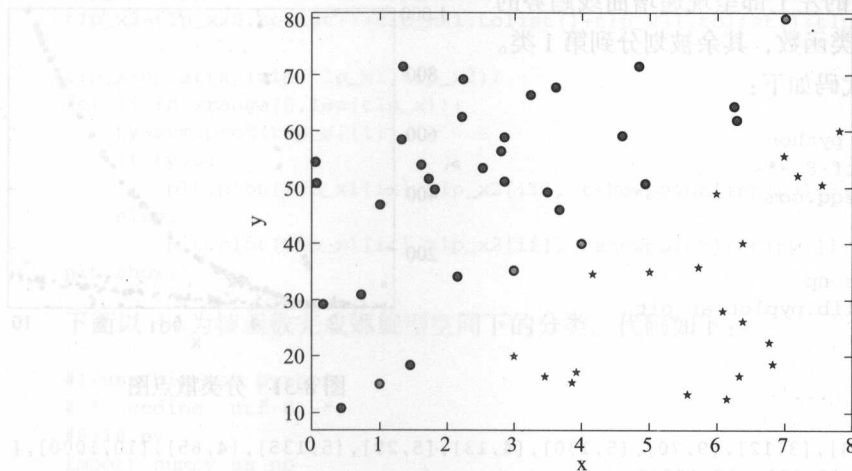


图 8-30 分类散点图

### 3. 非线性分类

非线性分类需要选择非线性核函数，这里选择 poly 作为 SVM 的核函数。尝试将分属于下面两个函数的坐标点分开：

第一类： $y=x^a+b$  ( $a \leq 2$ ,  $\text{abs}(b) < 10$ )

第二类： $y=x^a+b$  ( $a > 3$ ,  $\text{abs}(b) < 10$ )

1) 设置样本数据。代码如下：

```
x= [[1,1],[2,4],[3,12],[9,70],[5,130],[4,13],[5,29],[5,135],[4,68],[10,1000],[8,520],[7,340],[6,40],[10,150]]
y=[1,1,1,1,0,1,1,0,0,0,0,0,1,1]
```

2) 设置 SVM 的核函数，进行训练。

```
x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=10)
svm.learn(x, y)
```

3) 随机产生待分类数据，进行测试。代码如下：

```
tlp_x10=np.random.rand(100)*(xmax-xmin)+xmin
tlp_x20=tlp_x10**3+np.random.rand(100)*20-10
tlp_x11=np.random.rand(100)*(xmax-xmin)+xmin
tlp_x21=tlp_x11**2+np.random.rand(100)*20-10
tlp_x30=np.random.rand(50)*(xmax-xmin)+xmin
tlp_x31=tlp_x30**((round(np.random.rand()*6,0)+3)+np.random.rand(50)*10-5
```

```

t1p_x40=np.random.rand(50)*(xmax-xmin)+xmin
t1p_x41=t1p_x30**((round(np.random.rand(),0)+1)+np.random.rand(50)*10-5

```

如图 8-31 所示是程序生成的散点图，实心圆圈和星号分别代表不同类的数据。数据点被成功划分为两类：图的左上部呈现递增曲线趋势的数据点划分到了第 2 类函数，其余被划分到第 1 类。

完整的 Python 代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#email:myhaspl@qq.com
#author: 麦好
#8-15.py
import numpy as np
import matplotlib.pyplot as plt
import mlp

```

```

print 'loading ...'

```

```

x = [[1,1],[2,4],[3,12],[9,70],[5,130],[4,13],[5,29],[5,135],[4,65],[10,1000],[
8,520],[7,340],[6,40],[10,150]]
y=[1,1,1,1,0,1,1,0,0,0,0,0,1,1]

```

```

showpoint=['ro','r*']
tshowpoint=['bo','b*']

```

```

x=np.array(x)

```

```

y=np.array(y)

```

```

svm = mlp.LibSvm(svm_type='c_svc', kernel_type='poly', gamma=50)

```

```

svm.learn(x, y)

```

```

lp_x1 = x[:,0]

```

```

lp_x2 = x[:,1]

```

```

xmin, xmax = np.min(lp_x1)-0.5, np.max(lp_x1)+0.5

```

```

ymin, ymax = np.min(lp_x2)-0.5, np.max(lp_x2)+0.5

```

```

plt.subplot(111)

```

```

plt.xlabel(u"x")

```

```

plt.xlim(xmin, xmax)

```

```

plt.ylabel(u"y")

```

```

plt.ylim(ymin, ymax)

```

```

# 显示样本点

```

```

for ii in xrange(0,len(x)):

```

```

    ty=svm.pred(x[ii])

```

```

    if ty>0:

```

```

        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])

```

```

    else:

```

```

        plt.plot(lp_x1[ii], lp_x2[ii], showpoint[int(ty)])

```

```

# 未知样本分类

```

```

t1p_x10=np.random.rand(100)*(xmax-xmin)+xmin

```

```

t1p_x20=t1p_x10**3+np.random.rand(100)*20-10

```

```

t1p_x11=np.random.rand(100)*(xmax-xmin)+xmin

```

```

t1p_x21=t1p_x11**2+np.random.rand(100)*20-10

```

```

t1p_x30=np.random.rand(50)*(xmax-xmin)+xmin

```

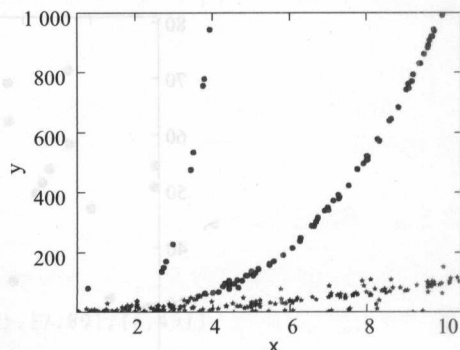


图 8-31 分类散点图

```

t1p_x31=t1p_x30**((round(np.random.rand()*6,0)+3)+np.random.rand(50)*10-5
t1p_x40=np.random.rand(50)*(xmax-xmin)+xmin
t1p_x41=t1p_x30**((round(np.random.rand(),0)+1)+np.random.rand(50)*10-5

t1p_x1=t1p_x10.tolist()+t1p_x11.tolist()+t1p_x30.tolist()+t1p_x40.tolist()
t1p_x2=t1p_x20.tolist()+t1p_x21.tolist()+t1p_x31.tolist()+t1p_x41.tolist()

t1p_x=np.array(zip(t1p_x1,t1p_x2))
for ii in xrange(0,len(t1p_x)):
    ty=svm.pred(t1p_x[ii])
    if ty>0:
        plt.plot(t1p_x1[ii],t1p_x2[ii], tshowpoint[int(ty)])
    else:
        plt.plot(t1p_x1[ii],t1p_x2[ii], tshowpoint[int(ty)])
plt.show()

```

下面以 rbf 为核函数完成螺旋型空间下的分类。代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#8-16.py
import numpy as np
import matplotlib.pyplot as plt
import mlpy
f = np.loadtxt("spiral.data")
x, y = f[:, :2], f[:, 2]
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='rbf', gamma=100)
svm.learn(x, y)
xmin, xmax = x[:,0].min()-0.1, x[:,0].max()+0.1
ymin, ymax = x[:,1].min()-0.1, x[:,1].max()+0.1
xx, yy = np.meshgrid(np.arange(xmin, xmax, 0.01), np.arange(ymin, ymax, 0.01))
xnew = np.c_[xx.ravel(), yy.ravel()]
ynew = svm.pred(xnew).reshape(xx.shape)
fig = plt.figure(1)
plt.pcolormesh(xx, yy, ynew)
plt.scatter(x[:,0], x[:,1], c=y)
plt.show()

```

如图 8-32 所示为分类的效果图。两类数据被螺旋形分界线划分。

## 8.6 回归算法

回归分析是统计学上分析数据的一种方法，目的在于了解两个或多个变量间是否相关，以及相关的方向与强度，并建立数学模型，以便观察特定变量来预测研究者感兴趣的变量，它建立了因变量与自变量之间的关系模型。

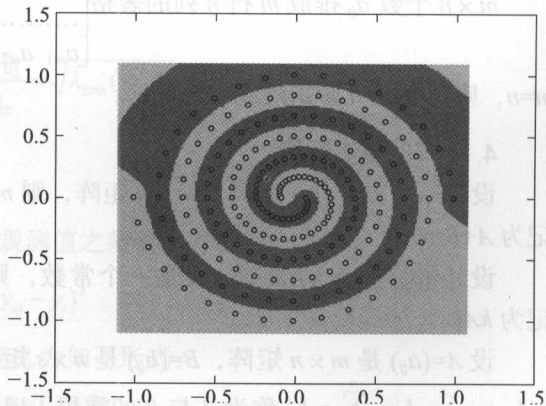


图 8-32 SVM 分类的效果图



## 8.6.1 线性代数基础

线性代数是数学的一个分支,它的研究对象是向量、向量空间(或称线性空间)、线性变换和有限维的线性方程组。向量空间是现代数学的一个重要课题,线性代数广泛地应用于抽象代数和泛函分析中,并能通过解析几何具体表示。线性代数在机器学习理论体系中的地位举足轻重。

### 1. 伴随矩阵

在线性代数中,一个方形矩阵的伴随矩阵是一个类似于逆矩阵的概念。如果矩阵可逆,那么它的逆矩阵和它的伴随矩阵之间只差一个系数。

矩阵  $A$  的伴随矩阵是  $A$  的余子矩阵的转置矩阵,具体定义如下:

$$\text{设 } A=(a_{ij})_{n \times n}, \text{ 则 } a_{i1}A_{j1}+a_{i2}A_{j2}+\cdots+a_{in}A_{jn}=\begin{cases} |A|, & i=j \\ 0, & i \neq j \end{cases}$$

$$\text{即 } AA^*=A^*A=|A|E$$

其中,

$$A^*=\begin{pmatrix} A_{11} & A_{21} & \cdots & A_{n1} \\ A_{12} & A_{22} & \cdots & A_{n2} \\ \cdots & \cdots & \cdots & \cdots \\ A_{1n} & A_{2n} & \cdots & A_{nn} \end{pmatrix}=(A_{ji})=(A_{ij})^T$$

若  $A$  可逆,则  $A$  的伴随矩阵  $A^*=|A|A^{-1}, (A^*)^*=\frac{1}{|A|}A$ 。

### 2. 方阵的行列式运算

设  $A, B$  为  $n$  阶方阵,则  $|AB|=|A||B|=|B||A|=|BA|$

但  $|A \pm B|=|A| \pm |B|$  不一定成立。

### 3. 矩阵与方阵

$m \times n$  个数  $a_{ij}$  排成  $m$  行  $n$  列的表格  $\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{12} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$  称为矩阵,简记为  $A$ ,或  $(a_{ij})_{m \times n}$ 。若

$m=n$ ,则称  $A$  是  $n$  阶矩阵或  $n$  阶方阵。

### 4. 矩阵的基本运算

设  $A=(a_{ij}), B=(b_{ij})$  是两个  $m \times n$  矩阵,则  $m \times n$  矩阵  $C=(c_{ij})=a_{ij}+b_{ij}$  称为矩阵  $A$  与  $B$  的和,记为  $A+B=C$ 。

设  $A=(a_{ij})$  是  $m \times n$  矩阵, $k$  是一个常数,则  $m \times n$  矩阵  $(ka_{ij})$  称为数  $k$  与矩阵  $A$  的数乘,记为  $kA$ 。

设  $A=(a_{ij})$  是  $m \times n$  矩阵, $B=(b_{ij})$  是  $n \times s$  矩阵,那么  $m \times s$  矩阵  $C=(c_{ij})$ ,其中, $c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\cdots+a_{in}b_{nj}=\sum_{k=1}^n a_{ik}b_{kj}$  称为  $A$  与  $B$  的乘积,记为  $C=AB$ 。

## 5. 矩阵的秩

若  $A$  为  $n$  阶方阵, 则  $r(A^*) = \begin{cases} n, r(A)=n \\ 1, r(A)=n-1 \\ 0, r(A)<n-1 \end{cases}$

## 6. 线性相关与线性无关

$\alpha_1, \alpha_2, \dots, \alpha_s$  线性相关  $\Leftrightarrow$  至少有一个向量可以用其余向量线性表示。

若  $\alpha_1, \alpha_2, \dots, \alpha_s$  线性无关,  $\alpha_1, \alpha_2, \dots, \alpha_s, \beta$  线性相关  $\Leftrightarrow \beta$  可以由  $\alpha_1, \alpha_2, \dots, \alpha_s$  唯一线性表示。

## 8.6.2 最小二乘法原理

### 1. 范数

在向量空间  $R^n(C^n)$  中, 设  $x=(x_1, x_2, \dots, x_n)^T$ , 向量范数定义如下:

$x$  的 2-范数或欧氏范数计算公式为:

$$\|x\|_2 = (|x_1|^2 + |x_2|^2 + \dots + |x_n|^2)^{1/2}$$

$x$  的 1-范数计算公式为:

$$\|x\|_1 = |x_1| + |x_2| + \dots + |x_n|$$

$x$  的  $\infty$  范数或最大范数计算公式为:

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

$x$  的  $p$  范数计算公式为:

$$\|x\|_p = (|x_1|^p + |x_2|^p + \dots + |x_n|^p)^{1/p}$$

$A$  的行范数计算公式如下:

$$\|A\|_\infty = \max_{x \neq 0} \frac{\|A\bar{x}\|_\infty}{\|\bar{x}\|_\infty} = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$$

$A$  的列范数计算公式如下:

$$\|A\|_1 = \max_{\bar{x} \neq 0} \frac{\|A\bar{x}\|_1}{\|\bar{x}\|_1} = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}|$$

$A$  的 2-范数或谱范数计算公式如下:

$$\|A\|_2 = \max_{\bar{x} \neq 0} \frac{\|A\bar{x}\|_2}{\|\bar{x}\|_2} = \sqrt{\lambda_{\max}(A^T A)}$$

其中  $\lambda_{\max}(A^T A)$  为  $A^T A$  的最大特征值。

### 2. 算法目标

线性回归算法通过适合的参数, 使函数与观测值之差的平方和最小, 即:

$$\min_{\bar{x}} \sum_{i=1}^n (y_m - y_i)^2$$

1) 二元线性回归。首先将线性回归函数定义为如下形式:

$$y = x_0 + x_1 t$$

8.6 根据其算法目标, 得到下式:

$$\min_{x_0, x_1} \left\| \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right\|_2 \min_x \|Ax - b\|_2$$

解上式, 得到如下形式:

$$x_1 = \frac{\sum_{i=1}^n (t_i - \bar{t})(y_i - \bar{y})}{\sum_{i=1}^n (t_i - \bar{t})^2}$$

$$x_0 = \bar{y} - x_1 \bar{t}$$

其中  $\bar{t} = \frac{1}{n} \sum_{i=1}^n t_i$ , 为  $t$  值的算术平均值。

2) 多元线性回归。将多个不相关变量  $t_1, \dots, t_q$ , 组成下面线性函数:

$$y(t_1, \dots, t_q; x_0, x_1, \dots, x_q) = x_0 + x_1 t_1 + \dots + x_q t_q$$

上式可写成  $Ax=b$  的形式, 如下所示:

$$\begin{pmatrix} 1 & t_{11} & \cdots & t_{1j} & \cdots & t_{1q} \\ 1 & t_{21} & \cdots & t_{2j} & \cdots & t_{2q} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & t_{i1} & \cdots & t_{ij} & \cdots & t_{iq} \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & t_{n1} & \cdots & t_{nj} & \cdots & t_{nq} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_q \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{pmatrix}$$

接着求解目标函数:

$$\min_x \|Ax - b\|_2, A \in C^{m \times n}, b \in C^m$$

其特解为  $A$  的广义逆矩阵与  $b$  的乘积, 这同时也是 2-范数极小的解, 其通解为特解加上  $A$  的零空间。

## 8.6.3 线性回归

### 1. 回归模型求解

上节讲述了一些多元线性回归模型, 可表示为  $Ax=b$  的形式, 求解回归模型就是求解  $A$  代表的参数值的估计值, 求解过程就是线性回归的过程。根据最小二乘原理及相关数学推导, 参数估计值为:

$$\tilde{B} = (X^T X)^{-1} X^T$$

### 2. 一元线性回归

如果随机变量  $y$  与变量  $x$  之间呈现某种线性关系, 则  $y$  与  $x$  之间一元线性回归模型为:

$$\hat{y} = a + bx$$

上式也称变量  $y$  对变量  $x$  的一元线性回归方程,  $a$ 、 $b$  称为回归系数。

根据前面讲述的最小二乘法原理，用 Python 编码实现。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-17.py

import matplotlib.pyplot as plt
x=[1,2,3,3,6,12,11]
y=[3,5,8,5,12,26,20]
average_x=float(sum(x))/len(x)
average_y=float(sum(y))/len(y)
x_sub=map((lambda x:x-average_x),x)
y_sub=map((lambda x:x-average_y),y)
x_sub_pow2=map((lambda x:x**2),x_sub)
y_sub_pow2=map((lambda x:x**2),y_sub)
x_y=map((lambda x,y:x*y),x_sub,y_sub)
a=float(sum(x_y))/sum(x_sub_pow2)
b=average_y-a*average_x
plt.xlabel('X')
plt.ylabel('Y')
plt.plot(x, y, '*')
plt.plot([0,15],[0*a+b,15*a+b])
plt.grid()
plt.title("{0}*x+{1}".format(a,b))
plt.show()
```

如图 8-33 所示为程序绘制的散点图，这些数据点呈明显的线性趋势，程序将它们的线性趋势绘制成一条回归线，回归效果不错。

### 3. 多元线性回归

一元线性回归是指将一个主要影响因素作为自变量，分析自变量的变化如何引起因变量的变化。在现实问题的研究中，因变量的变化往往受几个重要因素的影响，此时就需要用两个或两个以上的影响因素作为自变量来解释因变量的变化，这就是多元回归。

设  $y$  为因变量， $x_1, x_2, \dots, x_k$  为自变量，并且自变量与因变量之间为线性关系时，则多元线性回归模型为：

$$y=b_0+b_1x_1+b_2x_2+\dots+b_kx_k+e$$

根据最小二乘法，使用 Python 来实现这个回归模型。代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
```

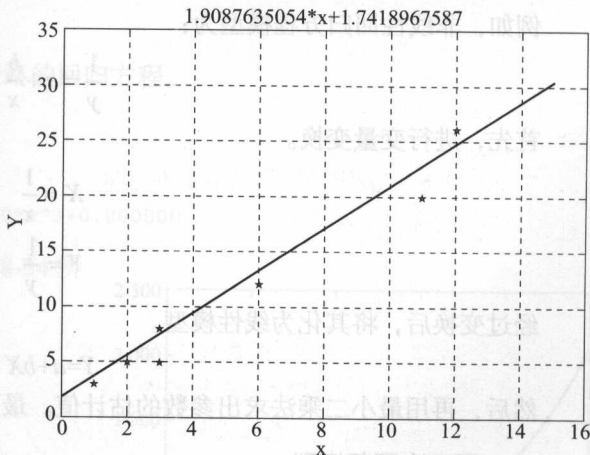


图 8-33 散点图

```
#8-18.py
import numpy as np
x = np.matrix([[7,2,3],[3,7,17],[11,3,5]],dtype=np.float64)
y = np.matrix([28,40,44],dtype=np.float64).T
b=(x.T*x).I*x.T*y
print u" 参数项矩阵为 {0}".format(b)
i=0
cb=[]
while i<3:
    cb.append(b[i,0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T
print "y=%f*x1+%f*x2+%f*x3+%f"%(cb[0],cb[1],cb[2],mye)
```

上述代码计算了参数估计值，并列出了多元线性回归方程。

参数项矩阵为 [[ 3.]

[ 2.]

[ 1.]]

y=3.000000\*x1+2.000000\*x2+1.000000\*x3+-0.000000

## 8.6.4 多元非线性回归

如果自变量  $X_1, X_2, \dots, X_m$  与因变量  $Y$  皆具有非线性关系，或者有的为非线性，有的为线性，则需要选择多元非线性回归模型。非线性回归方程很难求解，通常把非线性回归转化为线性回归，然后应用最小二乘法求解。

例如，非线性回归方程模型为：

$$\frac{1}{y} = a + \frac{b}{x}$$

首先，进行变量变换。

$$X = \frac{1}{x}$$

$$Y = \frac{1}{y}$$

经过变换后，将其化为线性模型。

$$Y = a + bX$$

然后，再用最小二乘法求出参数的估计值。最后经过适当的变换，得到所求回归曲线。

### 1. 一元三次回归模型

一元三次回归模为：

$$\hat{y}_3 = b_0 + b_1x + b_2x^2 + b_3x^3$$

用 Python 代码实现：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```



```

#code:myhaspl@qq.com
#8-19.py
#y=b1*x+b2*(x^2)+b3*(x^3)
import numpy as np
import matplotlib.pyplot as plt
z=np.matrix([3,1.4,1.9]).T
myx =np.matrix([[7],[3],[9]],dtype=np.float64)
x = np.matrix([[myx[0,0],myx[0,0]**2,myx[0,0]**3],\
               [myx[1,0],myx[1,0]**2,myx[1,0]**3],\
               [myx[2,0],myx[2,0]**2,myx[2,0]**3]],\
               dtype=np.float64)
y =x*z
b=(x.T*x).I*x.T*y
print u" 参数项矩阵为 {}".format(b)
i=0
cb=[]
while i<3:
    cb.append(b[i,0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T

print "y=%f*x+%f*x^2+%f*x^3+%f"%(cb[0],cb[1],cb[2],mye)

pltx=np.linspace(0,10,1000)
plty=cb[0]*pltx+cb[1]*(pltx**2)+cb[2]*(pltx**3)+mye
plt.plot(myx,y,"*")
plt.plot(pltx,plty)
plt.show()

```

程序运行后, 计算参数估计值以及最终的回归方程。

参数项矩阵为 [[ 3. ]

[ 1.4]

[ 1.9]]

$y=3.000000*x+1.400000*x^2+1.900000*x^3+0.000000$

非线性回归方程表现为曲线, 如图 8-34 所示是上述代码生成的回归曲线。

## 2. 二元二次多项式回归方程

二元二次多项式回归方程为:

$$\hat{y}=a+b_{11}x_1+b_{21}x_2+b_{12}x_1^2+b_{22}x_2^2+b_{11 \times 22}x_1x_2$$

$$\text{令 } b_1=b_{11}, b_2=b_{21}, b_3=b_{12}, b_4=b_{22}, b_5=b_{11 \times 22}$$

$$x_3=x_1^2, x_4=x_2^2, x_5=x_1 \cdot x_2$$

这样上式即可化为以下五元一次线性回归方程:

$$\hat{y}=a+b_1x_1+b_2x_2+b_3x_3+b_4x_4+b_5x_5$$

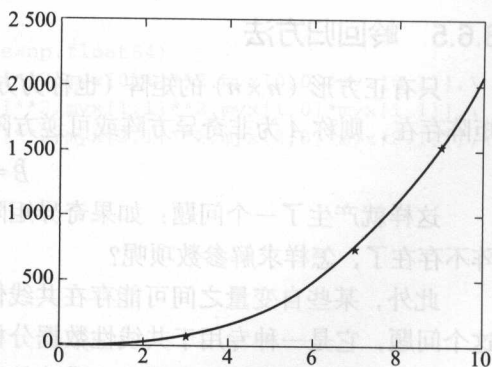


图 8-34 回归曲线

根据推导结果来看,可以按多元线性回归方法计算各偏回归系数,建立二元二次多项式回归方程。

下面用 Python 求解二元二次多项式回归方程。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com
#8-20.py
#y=b1*x1+b2*(x2^2)+b3*x1*x2
import numpy as np

z=np.matrix([3,1.4,1.9]).T
myx =np.matrix([[7,3],[3,17],[11,5]],dtype=np.float64)
x = np.matrix([[myx[0,0],myx[0,1]**2,myx[0,0]*myx[0,1]],\
               [myx[1,0],myx[1,1]**2,myx[1,0]*myx[1,1]],\
               [myx[2,0],myx[2,1]**2,myx[2,0]*myx[2,1]]],\
               dtype=np.float64)

y =x*z
b=(x.T*x).I*x.T*y
print u" 参数项矩阵为 {0}".format(b)
i=0
cb=[]
while i<3:
    cb.append(b[i,0])
    i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T

print "y=%f*x1+%f*x2^2+%f*x1*x2+%f"%(cb[0],cb[1],cb[2],mye)
```

程序计算回归方程的参数后,列出回归方程,如下所示:

```
参数项矩阵为 [[ 3. ]
 [ 1.4]
 [ 1.9]]
y=3.000000*x1+1.400000*x2^2+1.900000*x1*x2+-0.000000
```

## 8.6.5 岭回归方法

只有正方形( $n \times n$ )的矩阵(也称为方阵),才可能有但非必然有逆矩阵。若方阵 $A$ 的逆矩阵存在,则称 $A$ 为非奇异方阵或可逆方阵。回顾一下参数项的求解公式:

$$\hat{B}=(X'X)^{-1}X'Y$$

这样就产生了一个问题:如果奇异矩阵或非方阵的矩阵不存在逆矩阵,即上式中的逆矩阵不存在了,怎样求解参数项呢?

此外,某些自变量之间可能存在共线性,这也给求逆带来了麻烦。岭回归方法可以解决这个问题,它是一种专用于共线性数据分析的有偏估计回归方法,实质上它是一种改良的最小二乘估计法,是通过放弃最小二乘法的无偏性,以损失部分信息、降低精度为代价,获得

回归系数更为符合实际、更可靠的回归方法，对病态数据的耐受性远远强于最小二乘法。其基本原理是：

给参数项求解公式中的  $X'X$  部分加上正常数矩阵  $kI$  ( $k>0$ )，则  $X'X+kI$  接近奇异的程度较小，如果  $X'X+kI$  可逆，则参数的求解公式变为：

$$\hat{B}(k) = (X'X + kI)^{-1} X'Y$$

其中  $k$  称为岭参数。 $\hat{B}(k)$  作为  $B$  的估计应比最小二乘估计  $\hat{B}$  稳定，当  $k=0$  时，岭回归估计就是普通的最小二乘估计。因为岭参数  $k$  不是唯一确定的，所以得到的岭回归估计  $\hat{B}(k)$  实际是对回归参数  $B$  的估计值。

选择一个适合的岭参数很重要，岭迹法是方法之一。选择的原则是：

□ 各回归系数的岭估计基本稳定。

□ 系数符号变得合理。

□ 残差平方和。

□ 取使方程基本稳定的最小  $K$  值。

### 8.6.6 伪逆方法

除了岭回归方法，伪逆方法也是一种不错的方法，它是对逆阵的推广。一般所说的伪逆是指 Moore-Penrose 伪逆，它是由 E. H. Moore 和 Roger Penrose 分别独立提出的。

一个与  $A$  的转置矩阵  $A^T$  同型的矩阵  $X$ ，满足： $AXA=A$ ， $XAX=X$ 。此时，称矩阵  $X$  为矩阵  $A$  的伪逆，也称为广义逆矩阵。

numpy 提供了求伪逆的函数 `linalg.pinv()`，可调用 numpy 的这个方法对一组数据进行回归分析，这组数据不适合用前面介绍的非线性回归算法计算。代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# code: myhaspl@qq.com
# 8-21.py
# y=b1*x1+b2*x2+b3*(x1^2)+b4*(x2^2)+b5*x1*x2
import numpy as np

z=np.matrix([1.4,1.9,1.7,0.8,1.1]).T
myx =np.matrix([[7,3],[3,17],[11,5]],dtype=np.float64)
x = np.matrix([[myx[0,0],myx[0,1],myx[0,0]**2,myx[0,1]**2,myx[0,0]*myx[0,1]],\
               [myx[1,0],myx[1,1],myx[1,0]**2,myx[1,1]**2,myx[1,0]*myx[1,1]],\
               [myx[2,0],myx[2,1],myx[2,0]**2,myx[2,1]**2,myx[2,0]*myx[2,1]]],\
               dtype=np.float64)

y =x*z
wn=np.linalg.pinv(x.T*x)
b=wn*x.T*y
print u" 参数项矩阵为 {0}".format(b)
i=0
cb=[]
while i<5:
    cb.append(b[i,0])
```

```

i+=1
temp_e=y-x*b
mye=temp_e.sum()/temp_e.size
e=np.matrix([mye,mye,mye]).T

print "y=%f*x1+%f*x2+%f*(x1^2)+%f*(x2^2)+%f*x1*x2+%f"%(cb[0],cb[1],cb[2],cb[3],
cb[4],mye)

```

程序运行后，输出的参数矩阵及回归方程如下：

```

参数项矩阵为 [[ 1.59659657]
 [ 0.69002152]
 [ 2.01029166]
 [ 0.9820693 ]
 [ 0.4052783 ]]
y=1.596597*x1+0.690022*x2+2.010292*(x1^2)+0.982069*(x2^2)+0.405278*x1
*x2+-0.000000

```

## 8.7 PCA 降维

PCA 主要用于数据降维。由一系列特征组成的多维向量，其中某些元素本身没有区分性，比如某个元素在所有的样本中都相等，或者彼此差距不大，那么这个元素本身就没有区分性，如果用它做特征来区分，贡献会非常小。我们的目的是找到那些变化大的元素，即方差大的维，而去除掉那些变化不大的维。

使用 PCA 的好处在于，可以对新求出的“主元”向量的重要性进行排序。根据需要取前面最重要的部分，将后面的维数省去，从而达到降维、简化模型或对数据进行压缩的效果。同时最大程度地保持了原有数据的信息，较低的维数意味着运算量的减少，在数据较多的情况带来的性能提高更明显。

PCA 通过将主成分分析的问题转化为求解协方差矩阵的特征值和特征向量来计算。其目标是寻找  $r(r < n)$  个新变量，使它们反映事物的主要特征，压缩原有数据矩阵的规模，每个新变量是原有变量的线性组合，体现原有变量的综合效果，这  $r$  个新变量称为“主成分”，它们可以在很大程度上反映原来  $n$  个变量的影响，并且这些新变量是互不相关的，也是正交的。

以将数据的维数从  $N$  降到 5 为例，PCA 算法过程如下：

- 1) 计算样本矩阵  $X$  协方差矩阵。
- 2) 计算协方差矩阵  $S$  的特征向量  $e_1, e_2, \dots, e_N$  及其特征值  $i=1, 2, \dots, N$ 。
- 3) 把特征值按从大到小排序，取前 5 位特征值对应的特征向量组成投影矩阵  $W$ 。
- 4) 投影数据到  $W$  组成的空间之中。

上述算法过程较抽象，理解它的最好方式就是动手实现它。下面编写一段 Python 程序，使用 `mlpy` 库提供的 PCA 类，实现 PCA 算法。

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-
#code:myhaspl@qq.com

```

```

#8-22.py
import numpy as np
import matplotlib.pyplot as plt
import mlpy
np.random.seed(0)
mean, cov, n = [0, 0], [[1,1],[1,1.5]], 100
x = np.random.multivariate_normal(mean, cov, n)
pca = mlpy.PCA()
pca.learn(x)
coeff = pca.coeff()
fig = plt.figure(1)
plot1 = plt.plot(x[:, 0], x[:, 1], 'o')
plot2 = plt.plot([0,coeff[0, 0]], [0, coeff[1, 0]], linewidth=4, color='r')
plot3 = plt.plot([0,coeff[0, 1]], [0, coeff[1, 1]], linewidth=4, color='g')
xx = plt.xlim(-4, 4)
yy = plt.ylim(-4, 4)
z = pca.transform(x, k=1)
xnew = pca.transform_inv(z)
fig2 = plt.figure(2)
plot1 = plt.plot(xnew[:, 0], xnew[:, 1], 'o')
xx = plt.xlim(-4, 4)
yy = plt.ylim(-4, 4)
plt.show()

```

如图 8-35 所示是程序生成的两张散点图，左边是数据降维前的效果，右边是数据降维后的效果。能明显看出数据降维后，数据的整体趋势并没有改变，最大程度地保持了原有数据的信息。

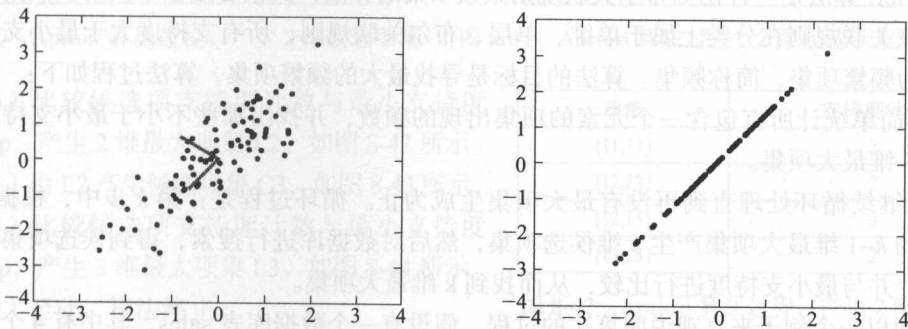


图 8-35 数据降维

## 8.8 关联规则

### 8.8.1 关联规则概述

以购买商品为例进行分析，假设某顾客同时购买了 A 商品和 B 商品，然后又购买了 C 商品，也就是说此次共买了商品 {A,B,C}，则可把 {A,B}、{C}、{A,B,C} 称为项集（购买项目的集合），可以建立一条规则，同时买了 A 和 B 两种商品的顾客可能会买 C 商品，记为



$\{A,B\} \rightarrow \{C\}$ ，这样通过一定数量的购买记录就能分析出用户的购买习惯。可将这一过程用形式化语言进行如下定义：

项集  $I$  由若干个项目组成，事务  $t$  由若干个项集组成，事务集  $T$  由若干事务  $t$  组成，设项集  $X$  是事务  $t_i \in T$  的一个子集，也可记为  $x \in t_i$ ，分析事务集，在指定支持度和置信度满足的情况下，认为规则  $X \rightarrow Y$  成立 ( $X$  和  $Y$  均为项集)，则称  $X$  为前件， $Y$  为后件。

支持度的计算公式如下：

$$\text{支持度} = T \text{ 中包括 } X \cup Y \text{ 的百分比} = \frac{\text{count}(X \cup Y)}{n}$$

其中  $n$  为事务集  $T$  的事务数量，count 表示数量。

置信度的计算公式如下：

$$\text{置信度} = T \text{ 中包括 } X \cup Y \text{ 的百分比} = \frac{\text{count}(X \cup Y)}{\text{count}(X)}$$

其中 count 表示数量。

置信度可理解为条件概率，即：在事务集中， $X$  项集存在的前提下，项集  $Y$  存在的可能性；支持度可理解为事务集  $T$  中  $X$  和  $Y$  项集同时存在的概率。可给支持度和置信度设置一个阈值，若这两个指标均大于这个阈值，则可生成关联规则。

## 8.8.2 频繁项集算法

### 1. Apriori 算法

Apriori 算法是一种挖掘布尔关联规则频繁项集的算法，其核心是基于两阶段频集思想的递推。该关联规则在分类上属于单维、单层、布尔关联规则，所有支持度大于最小支持度的项集称为频繁项集，简称频集。算法的目标是寻找最大的频繁项集，算法过程如下：

1) 简单统计所有包含一个元素的项集出现的频数，并找出那些不小于最小支持度的项集，即一维最大项集。

2) 继续循环处理直到再没有最大项集生成为止。循环过程是：第  $k$  步中，根据第  $k-1$  步生成的  $k-1$  维最大项集产生  $k$  维候选项集，然后对数据库进行搜索，得到候选项集的项集支持度，并与最小支持度进行比较，从而找到  $k$  维最大项集。

下面以一个例子来直观说明算法的过程。假设有一个数据库表 sales，其中有 4 个购买事务，如图 8-36 所示。

1) 将最小支持度 Minsup 设定为 2，算法的运行过程如图 8-37 所示。

顾客 ID	商品
1	I1,I3,I4
2	I2,I3,I5
3	I1,I2,I3,I5
4	I2,I5

顾客 ID	商品
1	I1,I3,I4
2	I2,I3,I5
3	I1,I2,I3,I5
4	I2,I5

图 8-36 数据库表 sales 的购买事务

图 8-37 Apriori 算法过程：设定最小支持度

2) 扫描上述 sales 表, 对每个候选项进行支持度计数, 得到表 C1, 如图 8-38 所示。

3) 比较候选项支持度计数与最小支持度 Minsup, 产生 1 维最大项集 L1, 如图 8-39 所示。

项集	支持度计数
{I1}	2
{I2}	3
{I3}	3
{I4}	1
{I5}	3

项集	支持度计数
{I1}	2
{I2}	3
{I3}	3
{I5}	3

图 8-38 Apriori 算法过程: 候选项支持度计数

图 8-39 Apriori 算法过程: 产生 1 维最大项集

4) 由 L1 产生候选项集 C2, 如图 8-40 所示。

5) 扫描 sales 表, 对每个候选项集进行支持度计数, 如图 8-41 所示。

项集
{I1,I2}
{I1,I3}
{I1,I5}
{I2,I3}
{I2,I5}
{I3,I5}

项集	支持度计数
{I1,I2}	1
{I1,I3}	2
{I1,I5}	1
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

图 8-40 Apriori 算法过程: 产生 2 维候选项集

图 8-41 Apriori 算法过程: 2 维候选项集支持度计数

6) 比较候选项支持度计数与最小支持度 Minsup, 产生 2 维最大项集 L2, 如图 8-42 所示。

7) 由 L2 产生候选项集 C3, 如图 8-43 所示。

8) 比较候选项支持度计数与最小支持度 Minsup, 产生 3 维最大项集 L3, 如图 8-44 所示。

9) 至此, 算法终止。

Apriori 算法的主要缺点如下:

□ 每一步都要产生候选项集, 循环产生的组合过多, 没有排除不应该参与组合的元素, 组合随着数据库记录的增加呈现出几何级数的增加。

□ 每次计算项集的支持度时, 都对事务集中的全部记录进行了一遍扫描, 从而增加了计算机系统的 I/O 开销。

项集	支持度计数
{I1,I3}	2
{I2,I3}	2
{I2,I5}	3
{I3,I5}	2

图 8-42 Apriori 算法过程: 产生 2 维最大项集

项集
{I2,I3,I5}

图 8-43 Apriori 算法过程: 产生 3 维候选项集

项集	支持度计数
{I2,I3,I5}	2

图 8-44 Apriori 算法过程: 产生 3 维最大项集

## 2. FP-growth 算法

针对 Apriori 算法的性能瓶颈问题——需要产生大量的候选项集和重复地扫描数据库，2000 年 Jiawei Han 等人提出了基于 FP 树生成频繁项集的 FP-growth 算法。该算法只需要进行两次数据库扫描且它不必使用候选集，直接将数据库压缩成一棵频繁模式树，最后通过这棵树生成关联规则。研究表明它比 Apriori 算法大约快一个数量级。

FP-growth 算法是一种不产生候选模式而采用频繁模式增长的方法挖掘频繁模式的算法。算法只需要两次数据扫描：

第一次扫描数据库，得到 1 维频繁项集。

第二次扫描数据库，利用 1 维频繁项集过滤数据库中的非频繁项，同时生成 FP 树。

由于 FP 树蕴涵了所有的频繁项集，其后频繁项集的挖掘只需要在 FP 树上进行即可。FP 树挖掘由以下两个阶段组成：

第一阶段建立 FP 树，即将数据库中的事务构造成一棵 FP 树。

第二阶段为挖掘 FP 树，即针对 FP 树挖掘频繁模式和关联规则。

FP-growth 算法可具体描述如下：

□ 输入：事务数据集 D，最小支持度 Minsup。

□ 输出：频繁模式的完全集。

具体方法如下。

首先，按如下方法构建 FP 树。

1) 扫描事务数据库，收集频繁项集 F 并统计支持度，对 F 按支持度降序排序，得到频率排序好的项表 L。

2) 创建 FP 树的根节点，用“null”标记它。对于 D 中的每个事务 T，执行：选择 T 中的频繁项，并按 L 中的顺序排序。设排序后的频繁项表为 [p|P]，其中 p 是第一个元素，而 P 是剩余元素的表。调用 insert\_tree([p|P], T)。该过程执行情况如下：

如果 T 有子女 N 使得  $N.itemName = p.itemName$ ，则 N 的计数增加 1；否则创建一个新节点 N，将其计数设置为 1，链接到它的父节点 T，并且通过节点链结构将其链接到具有相同 itemName 的节点。如果 P 非空，则递归地调用 insert\_tree(P, N)。

其次，通过 FP-growth(Tree,  $\alpha$ ) 函数来实现 FP 树的规则，初始调用 FP-growth(Tree, null) 的描述如下：

```

if Tree 含单个路径 P then {
  for 路径 P 中节点的每个组合 ( 记作  $\beta$  )
    产生模式  $\beta \cup \alpha$ ，其支持度为  $support = \beta$  中节点的最小支持度；
}
else for each  $\alpha_i$  在 Tree 的头部 do {
  产生模式  $\beta = \alpha_i \cup \alpha$ ，其支持度为  $support = \alpha_i.support$ ；
  构造  $\beta$  的条件模式，然后构造  $\beta$  的条件 FP 树 Tree $\beta$ ；
  if Tree $\beta \neq$  空集 then
    调用 FP_growth(Tree $\beta$ ,  $\beta$ )
  }
end
  
```

### 8.8.3 关联规则生成

#### 1. 关联规则生成算法

设  $f$  为一个频繁项集,  $a$  为后件,  $f-a$  为前件, 可将  $(f-a) \rightarrow a$  设为一条关联规则。如果  $(f-a)$  是关联规则, 则  $(f-a_{\text{sub}}) \rightarrow a_{\text{sub}}$  也是关联规则,  $a_{\text{sub}}$  为  $a$  的某个非空子集, 所有以  $a_{\text{sub}}$  为后件的规则都成为候选关联规则, 候选关联规则在支持度和置信度都大于指定阈值时, 可生成关联规则。

支持度的计算方式如下:

$$\text{sup} = \frac{\text{count}(f)}{n}$$

其中,  $n$  为事务集中事务的数量。

置信度的计算方式如下:

$$\text{conf} = \frac{\text{count}(f)}{\text{count}(f-a)}$$

当支持度  $\text{conf} >$  最小支持度  $\text{minconf}$ 、置信度  $\text{sup} >$  最小置信度  $\text{minsup}$  时, 可生成如下规则:

$$(f-a) \rightarrow a$$

产生频繁项集后, 就可计算关联规则了, 具体算法可采用如下方式。

第一步, 从频繁项集  $f$  中生成 1 项后件的候选关联规则, 并从中挑选  $\text{conf} > \text{minconf}$  和  $\text{sup} > \text{minsup}$  的候选规则作为关联规则。

第二步, 利用现有  $k-1$  项后件的关联规则, 生成  $k$  项后件的候选关联规则, 从中挑选  $\text{conf} > \text{minconf}$  和  $\text{sup} > \text{minsup}$  的候选规则作为关联规则。

#### 2. 关联规则生成例子

下面以图 8-36 所示的 sales 数据表中的频繁项集为例, 讲解如何使用最大频繁项集 L3 生成关联规则。设定  $\text{minconf}$  为 60%,  $\text{minsup}$  为 70%, 在此仅分析一次频繁项集 L3, 如图 8-45 所示。

首先, 从 1 项后件的候选关联规则开始。根据 L3 生成以下候选关联规则:

项集	支持度计数
{I2,I3,I5}	2

图 8-45 频繁项集 L3

(1)  $\{I2, I3\} \rightarrow \{I5\}$  [sup=2/4, conf=2/2]

(2)  $\{I2, I5\} \rightarrow \{I3\}$  [sup=2/4, conf=2/3]

(3)  $\{I3, I5\} \rightarrow \{I2\}$  [sup=2/4, conf=2/2]

根据  $\text{minconf}$  和  $\text{minsup}$  的限制, 选择规则 (2) 作为生成规则 H1, 如下所示。

$\{I2, I5\} \rightarrow \{I3\}$

然后, 根据 H1, 生成 2 项后件候选关联规则, 如下所示。

$\{I2\} \rightarrow \{I3, I5\}$  [sup=2/4, conf=2/3]

上述候选关联规则满足  $\text{minconf}$  和  $\text{minsup}$  的要求, 因此将它作为生成规则 H2。



最后，根据频繁项集 L3 生成以下两条关联规则：

H1:{I2,I5}->{I3}

H2:{I2}->{I3,I5}

8.8.4 实例分析

下面以分析购买记录为例，分别就 Apriori 算法和 FP-growth 算法讲解频繁项集和关联规则。

1. Apriori 算法

假设某超级市场的小吃和饮料信息如表 8-1 所示。

表 8-1 超级市场的小吃和饮料清单

0	'Chocolate'	'Cake'	8.95	'Food'
1	'Lemon'	'Cake'	8.95	'Food'
2	'Casino'	'Cake'	15.95	'Food'
3	'Opera'	'Cake'	15.95	'Food'
4	'Strawberry'	'Cake'	11.95	'Food'
5	'Truffle'	'Cake'	15.95	'Food'
6	Chocolate'	Eclair'	3.25	'Food'
7	'Coffee'	'Eclair'	3.5	'Food'
8	'Vanilla'	'Eclair'	3.25	'Food'
9	'Napoleon'	'Cake'	13.49	'Food'
10	'Almond'	Tart'	3.75	'Food'
11	'Apple'	'Pie'	5.25	'Food'
12	'Apple'	'Tart'	3.25	'Food'
13	'Apricot'	'Tart'	3.25	'Food'
...	...	...	...	...
47	'Vanilla'	'Frappuccino'	3.85	'Drink'
48	'Cherry'	'Soda'	1.29	'Drink'
49	'Single'	'Espresso'	1.85	'Drink'

表 8-1 的第 1 列表示食品编号 ID，之后的列依次是口味、品种、价格、类别。

商品购买记录如表 8-2 所示。

表 8-2 小吃和饮料的购买记录

1	7	15	44	49		
2	1	19				
3	1	19				
4	3	4	15	18	35	44
5	2	4	7	9	23	



(续)

6	14	21	44			
7	4	12	31	36	44	48
8	15	27	28			
9	2	28				
10	3	18	35			
11	23	24	40	41	43	
12	20	43	48			
13	49					
14	1	19	26			

表 8-2 中第一列为订单编号 ID，以后各列依次为每个订单所点的食品 ID。比如第一个订单的记录为“1,7,15,44,49”，表示订单 1 购买了 Coffee Éclair、a Blackberry Tart、Bottled Water 和 Single Espresso。

首先，下载 Apriori 程序包并解压。本书源码包中附带了 apriori.py 程序以及相关数据，也可以直接去 github.com 下载：<https://github.com/BastinRobin/apriori-python>。本书源码包的下载地址见前言。

然后，打开命令行，以最小支持度 3% 及最小置信度 70% 为标准，分析顾客的食品消费习惯：

```
python apriori.py data/1000/1000-out1.csv .03 .7
```

运行结果如下：

```
Dataset: data/1000/1000-out1.csv MinSup: 0.03 MinConf: 0.7
=====
1 : Berry Tart (14), Bottled Water (44) support= 0.034
2 : Strawberry Cake (4), Napoleon Cake (9) support= 0.049
3 : Chocolate Cake (0), Casino Cake (2) support= 0.04
4 : Raspberry Cookie (23), Lemon Lemonade (40) support= 0.031
5 : Marzipan Cookie (27), Tuile Cookie (28) support= 0.053
6 : Blueberry Tart (16), Apricot Croissant (32) support= 0.04
7 : Blueberry Tart (16), Hot Coffee (45) support= 0.033
8 : Gongolais Cookie (22), Truffle Cake (5) support= 0.058
9 : Cherry Tart (18), Opera Cake (3) support= 0.041
10 : Cheese Croissant (33), Orange Juice (42) support= 0.038
11 : Raspberry Cookie (23), Lemon Cookie (24) support= 0.033
12 : Lemon Cookie (24), Lemon Lemonade (40) support= 0.031
13 : Apricot Croissant (32), Hot Coffee (45), Blueberry Tart (16)
support= 0.032
14 : Apple Croissant (31), Apple Tart (12), Apple Danish (36),
Cherry Soda (48) support= 0.031

Skyline Itemsets: 14

Rule 1 : Apricot Croissant (32), Hot Coffee (45) --> Blue
berry Tart (16) [sup= 0.032 conf= 1.0 ]
```

Rule 2 :	Apple Croissant (31), --> Cherry Soda (48)	Apple Tart (12), [sup= 0.031	Apple Danish (36) conf= 0.775 ]
Rule 3 :	Apple Croissant (31), --> Apple Tart (12)	Apple Danish (36), [sup= 0.031	Cherry Soda (48) conf= 1.0 ]
Rule 4 :	Apple Tart (12), --> Apple Croissant (31)	Apple Danish (36), [sup= 0.031	Cherry Soda (48) conf= 1.0 ]

程序的上半部分为频繁项集，下半部分为关联规则。

首先，观察这 14 个频繁项集，第 1 个频繁项集表明 ID 号为 14 的 Berry Tart 经常和 ID 号为 44 的 Bottled Water 被顾客一起购买（有 0.034 的支持度），而拥有最大支持度的是第 8 个频繁项集，即：Gongolais Cookie 和 Truffle Cake 是顾客相对其他食品组合而言最喜欢一起购买的。

然后，观察关联规则，第 1 条关联规则是点了 Apricot Croissant 和 Hot Coffee 的顾客一般会来上一个 Blue berry Tart，这条关联规则的置信度为 1.0，说明顾客常有该规则表明的消费习惯。而置信度最小（为 77.5%）的是第 2 条规则，点 Apple Croissant、Apple Tart 及 Apple Danish 的顾客会选择 Cherry Soda，这个消费习惯可能会成为这部分顾客的选择。

2. FP-growth 算法

首先，下载 FP-growth 程序包并解压，本书源码包中提供有该程序包，也可以直接在 github.com 下载：<https://github.com/enaeseth/python-fp-growth>。该算法包需要安装，安装命令如下：

```
python setup.py install
```

然后，可直接运行程序 fp\_growth，该程序将自动分析其中的一个简单的数据集 tsk.csv，生成频繁项集。

tsk 数据集集中的数据如表 8-3 所示。

运行程序的命令如下：

```
python -m fp_growth -s 4 examples/tsk.csv
```

程序运行完毕后，输出支持数 support ≥ 4 的频繁项集，结果如下：

```
{a} 8
{c} 6
{a, c} 4
{b, c} 5
{b} 7
{a, b} 5
{d} 5
{a, d} 4
```

表 8-3 tsk 数据集

a	b		
b	c	d	
a	c	d	e
a	d	e	
a	b	c	
a	b	c	d
a			
a	b	c	
a	b	d	
b	c	e	

fp\_growth 程序的调用格式如下：

```
python -m fp_growth -s {minimum support} {path to CSV file}
```

其中, minimum support 为最小支持度, path to CSV file 为数据集的 csv 文件名。

最后, 以某网上商场的购买数据 sales.csv (共 1000 条记录, 本书附带的源码包中提供了该文件) 为例, 计算支持数 (支持数量)  $\geq 200$  的频繁项集。sales.csv 文件共有 1000 条记录, 记载了每笔订单的商品 ID 号, 内容如下所示。

```
2,5,27,29
15,27,31,33
9,16,22,
12,14,16,
2,9,44,
9,26,39,
10,31,,
1,47,49,
12,14,16,
12,14,16,
1,19,25,49
15,23,36,
15,17,36,
15,18,36,37
1,46,48,49
1,49,,
12,14,16,
9,22,26,
12,14,15,16
2,24,47,
9,22,23,
3,15,21,47
21,32,41,48
.....
```

编写程序 8-23.py, 调用 fp\_growth 程序库函数, 代码如下:

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
#8-23.py
from fp_growth import find_frequent_itemsets
import csv
myminsup=200# 设定最小支持度
if __name__ == '__main__':
    f = open("sales.csv")
    try:
        for itemset, support in find_frequent_itemsets(csv.reader(f), myminsup, True):
            print '(' + ','.join(itemset) + ')' + str(support)
    finally:
        f.close()
```

运行程序, 输出支持数  $\geq 200$  的频繁项集, 结果如下所示:

```
{9} 224
{12} 300
{14} 291
{12,14} 266
```

```
{22} 203
{16} 274
{12,16} 250
{14,16} 249
{12,14,16} 248
```

分析上面的频繁项集，{12}300 在 1 项组合中的支持数最多，表示很多顾客都喜欢购买 12 号商品；{12,14}266 在 2 项组合中的支持数最多，表示 12 和 14 号商品是较常见的 2 项购买组合，{12,14,16}248 在 3 项组合中的支持数最多，表示 12、14 和 16 号商品是较常见的 3 项购买组合。

程序 8-23.py 读取的 sales.csv 文件格式与在 github.com 下载的 fp\_growth 算法包所使用的 CSV 文件格式有所不同，因此，需要修改一下源码包中的 fp\_growth.py，使之兼容 sales.csv 和 tsk.csv 这两种文件的格式。修改方式为：在 fp\_growth.py 文件的 find\_frequent\_itemsets 函数中增加对空项目的判断，代码如下所示：

```
for transaction in transactions:
    processed = []
    for item in transaction:
        if item.strip()!='':
            items[item] += 1
            processed.append(item)
    processed_transactions.append(processed)
```

修改完毕后，需要重新安装 fp\_growth 包，安装命令如下：

```
python setup.py install
```

## 8.9 自动分类

自动分类是指由计算机系统按照被考察对象的内部或外部特征，根据一定的分类标准或分类参考（如类别的数量限制，同类对象的亲近程度等），将相近、相似或相同特征的对象聚合在一起或划分到不同种类的过程。

### 8.9.1 聚类算法

聚类分析又称群分析，它是研究（样品或指标）分类问题的一种统计分析方法，同时也是数据挖掘的一个重要算法。它以相似性为基础，相比于不同聚类的模式，同一聚类中的模式之间具有更多的相似性，聚类所要求划分的类是未知的。聚类算法应用相当广泛，已经被广泛用于考古学、地质勘探调查、天气预报、作物品种分类、土壤分类、微生物分类、经济管理、社会经济统计等应用中。通俗地说，聚类就是把相似的对象通过静态分类的方法分成不同的组别或更多的子集，让在同一个子集中的成员对象都有一些相似的属性。

#### 1. 系统聚类

系统聚类法分析的对象是大量的样本，可合理地对所有样品进行分类，同时没有任何模

式可供参考或依循,是在没有先验知识的情况下进行的。系统聚类法的原理是:同类事物具有很强的相似性,即同类事物之间的距离应该很小,可用距离统计量作为分类依据。具体算法思想如下:

首先假定各个样本各自成一类,这时各类间的距离就是各样品之间的距离,将距离最近的两类合并成一个新的类;然后计算新类与其他类间的距离,并将距离最近的两类合并,如此每次缩小一类,直至所有的样本都成为一类为止;最后根据需要或根据给出的距离临界值(或阈值)确定分类数及最终要分的类。其中的关键是距离计算,可采用如下距离函数来计算:

- 最短距离法,定义类与类之间的距离为两类最近样本间的距离。
- 最长距离法,定义类与类之间的距离为两类最远样本间的距离。
- 中间距离法,定义类与类之间的距离为两类中间样本间的距离。
- 类平均法,定义类与类之间的距离为两类样本对之间的平均距离或两类样本对之间平均距离的平均值。
- Mcquitty 相似法,与类平均法类似,但在类平均法的基础上增加了加权方法。
- 重心法,定义类与类之间的距离为两类重心(平均值)之间的距离。
- 离差平方和法,如果分类正确,同类样本之间的离差平方和应较小,不同样本之间的离差平方和应较大。

下面以某类型的 13 种商品的平均月销量为例,按销售数量分级聚类。销售数量如表 8-4 所示。

表 8-4 某类型商品的销售数量

ID	1	2	3	4	5	6	7	8	9	10	11	12	13
销量	500	600	200	12	38	59	482	295	260	279	410	552	677

可使用 R 语言的 hclust 函数进行聚类, R 语言代码如下:

```
> buy<-c(500,600,200,12,38,59,482,295,260,279,410,552,677)
> dim(buy)<-c(13,1)
> buy
      [,1]
[1,] 500
[2,] 600
[3,] 200
[4,] 12
[5,] 38
[6,] 59
[7,] 482
[8,] 295
[9,] 260
[10,] 279
[11,] 410
[12,] 552
```



```
[13,] 677
> d<-dist(buy)
# 最短距离法
> hclust(d,"single")->dshort
# 最长距离法
> hclust(d,"complete")->dlong
# 中间距离法
> hclust(d,"median")->dmedian
#Mcquitty 相似法
> hclust(d,"mcquitty")->dmcquitty
# 类平均法
> hclust(d,"average")->daverage
# 重心法
> hclust(d,"centroid")->dcentroid
# 离差平方和法
> hclust(d,"ward.D")->dward
# 画聚类树形图(谱系图)
> plot(dshort,hang=-1)
> plot(dlong,hang=-1)
> plot(dmedian,hang=-1)
> plot(dmcquitty,hang=-1)
> plot(daverage,hang=-1)
> plot(dcentroid,hang=-1)
> plot(dward,hang=-1)
```

上述代码的最后是用 plot 函数绘制所有聚类的谱系图,如图 8-46 至图 8-52 所示。

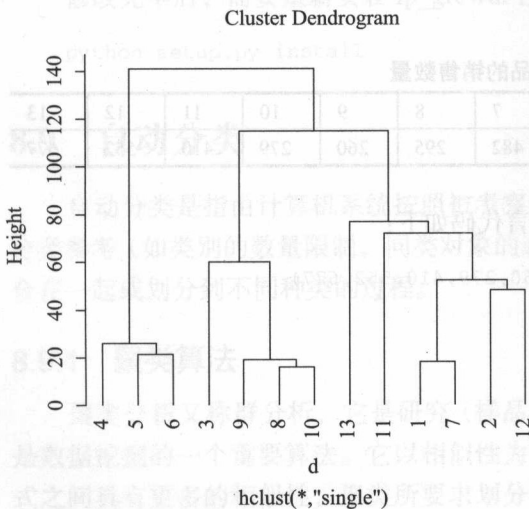


图 8-46 最短距离法

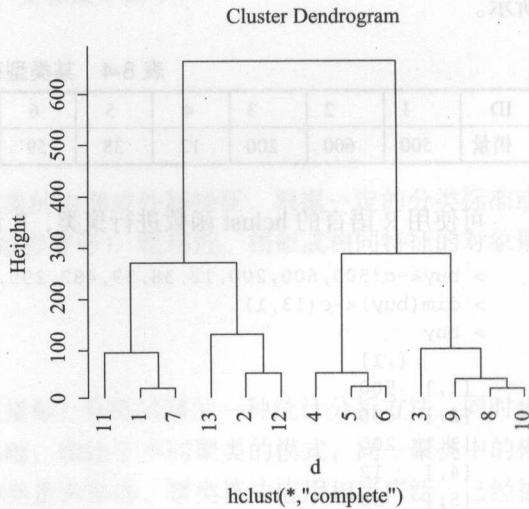


图 8-47 最长距离法

分析图 8-52 所示的谱系图,以离差平方和法为例,观察聚类分析的结果:按这 13 种商品的平均月销量进行聚类,1、7、11 号商品归为一组,而 12、2、13 号商品为一组,4、5、6 号商品可归为一组,3、8、9、10 号商品可归为一组。

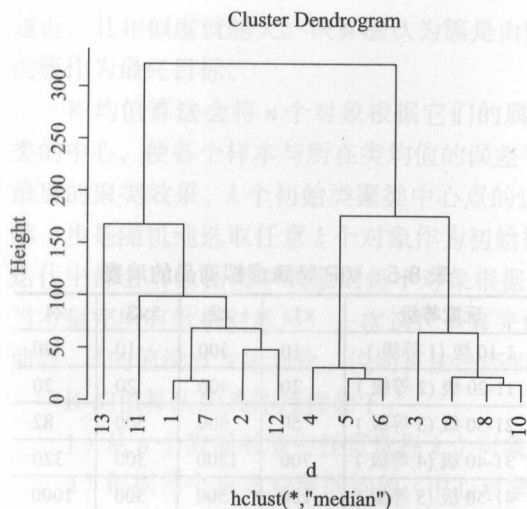


图 8-48 中间距离法

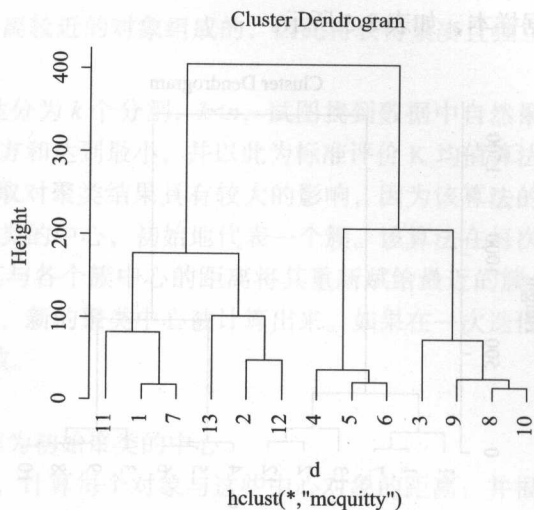


图 8-49 Mcquitty 相似法

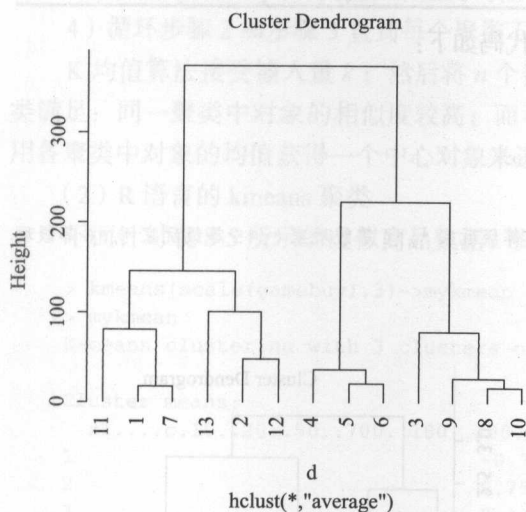


图 8-50 类平均法

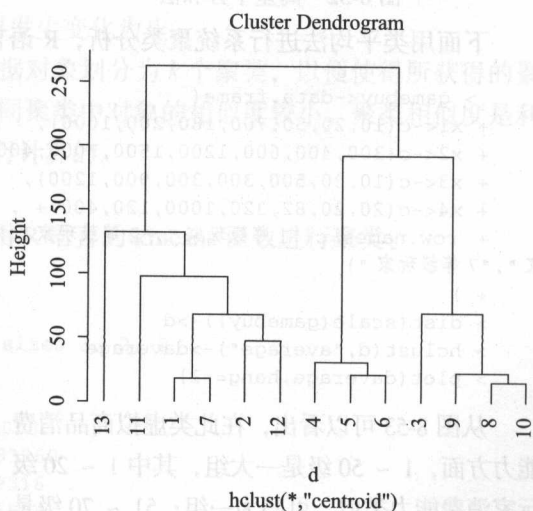


图 8-51 重心法

再来看一个例子，假设某游戏公司打算针对某游戏服务器内 1 级到 70 级之间的玩家按对某些 VIP 特殊虚拟商品的消费能力进行聚类，分别有以下 4 个指标。

- x1: 外部时装
- x2: 防具、武器、饰品及相关加强配料
- x3: 宠物及相关加强配件
- x4: 交通工具

然后，将每 10 级玩家设定为一个等级区（共有 7 级），随机抽取若干个该等级区内的典型玩家样本，计算在该等级区内平均每人每月消费的金额（游戏币）作为指标，生成以下数

据样本，如表 8-5 所示。

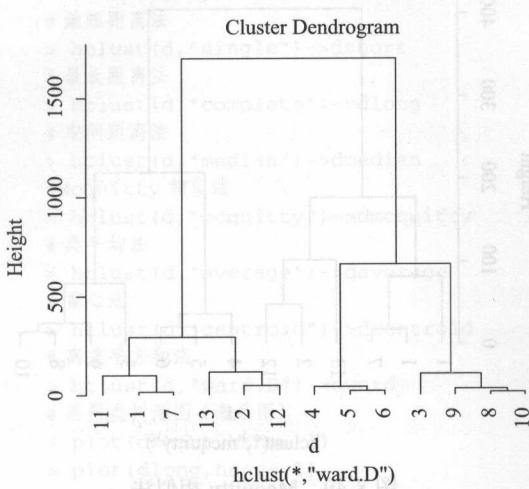


图 8-52 离差平方和法

表 8-5 VIP 特殊虚拟商品的消费

玩家等级	x1	x2	x3	x4
1-10 级 (1 等级)	10	300	10	20
11-20 级 (2 等级)	20	400	20	20
21-30 级 (3 等级)	50	600	500	82
31-40 级 (4 等级)	700	1200	300	320
41-50 级 (5 等级)	180	1500	300	1000
51-60 级 (6 等级)	200	3000	900	120
61-70 级 (7 等级)	1000	4000	1200	400

下面用类平均法进行系统聚类分析，R 语言代码如下：

```
> gamebuy<-data.frame(  
+ x1<-c(10,20,50,700,180,200,1000),  
+ x2<-c(300,400,600,1200,1500,3000,4000),  
+ x3<-c(10,20,500,300,300,900,1200),  
+ x4<-c(20,20,82,320,1000,120,400)+ ,  
+ row.names=c("1 等级玩家","2 等级玩家","3 等级玩家","4 等级玩家","5 等级玩家","6 等级玩  
家","7 等级玩家")  
+ )  
> dist(scale(gamebuy))->d  
> hclust(d,"average")->daverage  
> plot(daverage,hang=-1)
```

从图 8-53 可以看出，在此类虚拟商品消费能力方面，1 ~ 50 级是一大组，其中 1 ~ 20 级玩家消费能力类似，可归为一组；51 ~ 70 级是另一大组，玩家以 50 级为一个明显的界限。

## 2. K 均值算法

### (1) K 均值算法概述

系统聚类法一次成型以后就不能再改变，计算量较大，而 K 均值 (K-means) 算法属于动态聚类法，具有计算量较小、占计算机内存较少和方法简单的优点。K 均值算法是很典型的基于距离的聚类算法，采用距离作为相似性的评价指标，即认为两个对象的距离

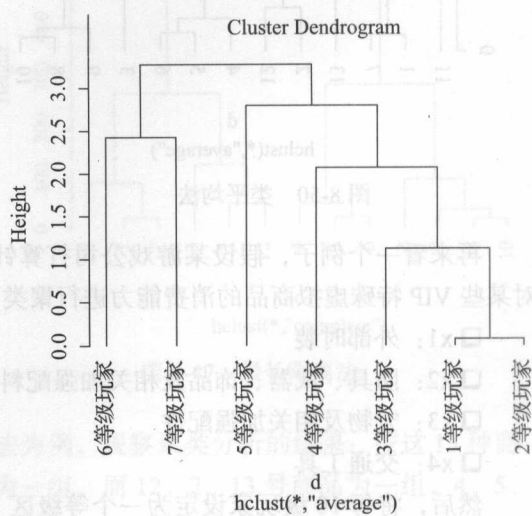


图 8-53 类平均法

越近,其相似度就越大。该算法认为簇是由距离较近的对象组成的,因此将获得紧凑且独立的簇作为最终目标。

K 均值算法会将  $n$  个对象根据它们的属性分为  $k$  个分割,  $k < n$ , 试图找到数据中自然聚类的中心,使各个样本与所在类均值的误差平方和达到最小,并以此为标准评价 K 均值算法最后的聚类效果。 $k$  个初始类聚类中心点的选取对聚类结果具有较大的影响,因为该算法的第一步是随机地选取任意  $k$  个对象作为初始聚类的中心,初始地代表一个簇。该算法在每次迭代中都会对数据集中剩余的每个对象根据其与各个簇中心的距离将其重新赋给最近的簇。当考察完所有数据对象后,一次迭代运算完成,新的聚类中心被计算出来。如果在一次迭代前后,  $J$  的值没有发生变化,说明算法已经收敛。

K 均值算法的具体过程如下:

- 1) 从  $n$  个数据对象中任意选择  $k$  个对象作为初始聚类的中心。
- 2) 根据每个聚类对象的均值(中心对象),计算每个对象与这些中心对象的距离;并根据最小距离重新对相应的对象进行划分。
- 3) 重新计算每个(有变化)聚类的均值(中心对象)。
- 4) 循环步骤 2 和步骤 3 直到每个聚类不再发生变化为止。

K 均值算法接受输入量  $k$ ; 然后将  $n$  个数据对象划分为  $k$  个聚类,以便使得所获得的聚类满足:同一聚类中对象的相似度较高;而不同聚类中对象的相似度较小。聚类相似度是利用各聚类中对象的均值获得一个中心对象来进行计算的。

## (2) R 语言的 kmeans 聚类

下面针对表 8-5 所示的虚拟商品数据,调用 R 语言的 kmeans 函数进行聚类:

```
> kmeans(scale(gamebuy),3)->mykmean
> mykmean
K-means clustering with 3 clusters of sizes 3, 2, 2

Cluster means:
x1....c.10..20..50..700..180..200..1000.
1                      -0.7283260
2                      0.7529316
3                      0.3395574
x2....c.300..400..600..1200..1500..3000..4000.
1                      -0.8042763
2                      1.3628949
3                      -0.1564805
x3....c.10..20..500..300..300..900..1200.
1                      -0.6393938
2                      1.3215563
3                      -0.3624657
x4....c.20..20..82..320..1000..120..400.
1                      -0.68490386
2                      -0.05798272
3                      1.08533851

Clustering vector:
```



1 等级玩家 2 等级玩家 3 等级玩家 4 等级玩家 5 等级玩家 6 等级玩家 7 等级玩家

1 1 1 3 3 2 2

Within cluster sum of squares by cluster:

[1] 0.8408947 2.9328150 2.8138040

(between\_SS / total\_SS = 72.6 %)

Available components:

[1] "cluster" "centers" "totss" "withinss"

[5] "tot.withinss" "betweenss" "size" "iter"

[9] "ifault"

> sort(mykmean\$cluster)

1 等级玩家 2 等级玩家 3 等级玩家 6 等级玩家 7 等级玩家 4 等级玩家 5 等级玩家

1 1 1 2 2 3 3

从以上分析结果可以看出，1、2、3 等级（1 ~ 30 级）玩家是一组，4、5 等级（31 ~ 50 级）玩家是一组，6、7 等级（51 ~ 70 级）玩家是一组。

### （3）Python 库 milk 的 kmeans 聚类

再来看一个例子，某网站对用户随机抽样，以抽取用户每周登录的次数作为用户对本网站热忱程度的分类标准，共随机抽取了 6 位用户，登录次数分别为：12、24、67、90、34、120，在此使用 Python 来完成这个任务，通过调用 milk 库函数实现 K-means 分类，代码如下：

# 样本矩阵声明

```
>>myx= np.array([12,24,67,90,34,120])
```

```
>>myx.shape=(6,1)
```

# 建立 k-means 分类器，第二个参数 3 表示分成 3 类。

```
>>learner =milk.kmeans(myx,3)
```

```
>>learner
```

```
(array([1, 1, 2, 0, 1, 0]), array([[ 105.      ], [ 23.33333333], [ 67.      ]]))
```

最后一行程序返回了两个结果，第 1 个结果是分类完成后样本所属的类别，第 2 个结果是每个类别的重心，观察结果可发现，第 1、2、5 号用户属于一类，重心为 105；而 3 号用户属于第 2 类，重心为 23.33333333；4、6 号用户属于第 3 类，重心为 67。

milk 是一种可供 Python 调用的机器学习工具包。在 Linux 环境中，需要首先下载 milk，网址如下：

<https://pypi.python.org/pypi/milk/>

下载完毕后，解压并进行安装，安装命令如下：

```
python setup.py install
```

在 Windows 环境中，可直接下载安装包进行安装，下载时请选择与 Python 版本对应的文件。下载地址如下：

<http://www.lfd.uci.edu/~gohlke/pythonlibs/#milk>

此外，有关 milk 的相关文档和资料可在 <http://luispedro.org/software/milk> 中找到。

### （4）Python 库 mlpy 的 kmeans 聚类

以下代码（程序 8-24.py）调用 mlpy 库的 kmeans 函数对若干个随机数进行聚类，并生成



效果图 (如图 8-54 所示)。

```
#8-24.py
import numpy as np
import matplotlib.pyplot as plt
import mlp
np.random.seed(0)
mean1, cov1, n1 = [1, 5], [[1,1],[1,2]], 200 # 200 points, mean=(1,5)
x1 = np.random.multivariate_normal(mean1, cov1, n1)
mean2, cov2, n2 = [2.5, 2.5], [[1,0],[0,1]], 300 # 300 points, mean=(2.5,2.5)
x2 = np.random.multivariate_normal(mean2, cov2, n2)
mean3, cov3, n3 = [5, 8], [[0.5,0],[0,0.5]], 200 # 200 points, mean=(5,8)
x3 = np.random.multivariate_normal(mean3, cov3, n3)
x = np.concatenate((x1, x2, x3), axis=0) # concatenate the samples
cls, means, steps = mlp.kmeans(x, k=3, plus=True)
fig = plt.figure(1)
plot1 = plt.scatter(x[:,0], x[:,1], c=cls, alpha=0.75)
plot2 = plt.scatter(means[:,0], means[:,1], c=np.unique(cls), s=128,
marker='d')
plt.show()
```

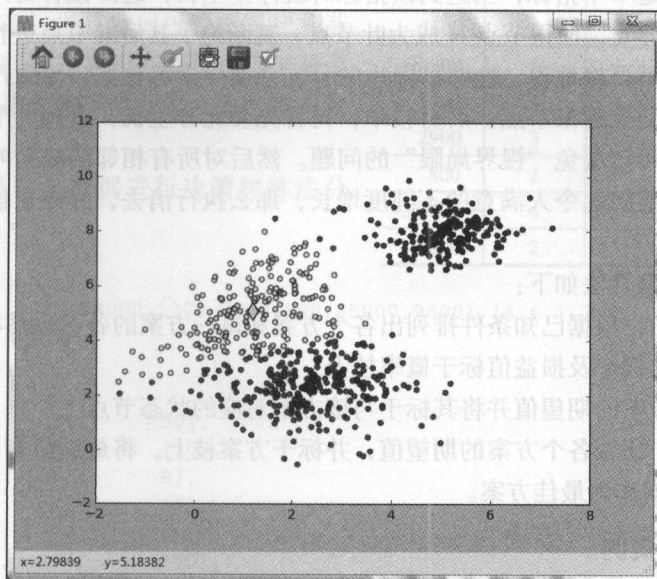


图 8-54 kmeans 聚类效果图

## 8.9.2 决策树

### 1. 决策树概述

决策树是一个预测模型，代表了对象属性与对象值之间的一种映射关系。树中的每个节点表示某个对象，而每个分叉路径则代表某个可能的属性值，每个叶节点则对应从根节点到该叶节点的路径所表示的对象的值。决策树仅有单一输出，若欲有复数输出，则可以建立独立的决策树以处理不同的输出。在数据挖掘中决策树是一种经常要用到的技术，可用于分析

数据,同样也可用来预测。图 8-55 展示了一个决策树。

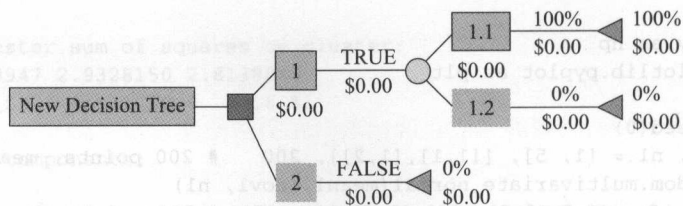


图 8-55 决策树

通常来说,决策树包含以下三种类型的节点。

□ 决策节点:通常用矩形框来表示。

□ 机会节点:通常用圆圈来表示。

□ 终节点:通常用三角形来表示。

剪枝是决策树停止分支的方法之一,剪枝又分预先剪枝和后剪枝两种。预先剪枝是在树的生长过程中设定一个指标,当达到该指标时就停止生长,这样做容易产生“视界局限”,也就是说一旦停止分支,使得节点 N 成为叶节点,就断绝了其后续节点进行“好”的分支操作的任何可能性。不严格地说,这些已停止的分支会误导学习算法,导致产生的树的不纯度降差最大的地方过分靠近根节点。后剪枝中,树首先要充分生长,直到叶节点都有最小的不纯度值为止,这样可以避免“视界局限”的问题。然后对所有相邻的成对叶节点考虑是否消去它们,如果消去能引起令人满意的不纯度增长,那么执行消去,并令它们的公共父节点成为新的叶节点。

决策树法的决策算法如下:

- 1) 绘制树状图,根据已知条件排列出各个方案和每一方案的各种自然状态。
- 2) 将各状态的概率及损益值标于概率枝上。
- 3) 计算各个方案的期望值并将其标于与该方案对应的状态节点上。
- 4) 进行剪枝,比较各个方案的期望值,并标于方案枝上,将最后所剩的期望值小的(即劣等方案剪掉)方案作为最佳方案。

## 2. 决策树算法实例

下面以实例来说明决策树算法,首先从最简单的 1 个属性的数据开始。假设某游戏公司在其官网的某游戏论坛中随机抽取了 6 位用户的资料为样本,以玩家平均每月所发游戏截图的数量为特征,进行决策树训练,训练好的决策树模型可将其他玩家分为热心玩家和非热心玩家。这 6 位用户平均每月所发游戏截图的数量为:12、24、67、90、34、120。

首先,训练决策树,Python 代码如下:

```
>>>features = np.array([12,24,67,90,34,120]) # 样本矩阵数据建立
>>>features.shape=(6,1)
>>>labels=np.array([False,False,False,True,False,True]) # 样本标签,False为非热心玩
家,True为热心玩家。
```

```
>> learner=milk.supervised.tree_learner() # 建立决策树分类器
>> model=learner.train(features, labels) # 样本数据训练, 返回分类器模型
```

然后, 对其他 8 位玩家的数据进行分类, Python 代码如下:

```
>> testx=np.array([10,20,40,90,33,89,199,200]) # 建立测试样本数据, 即测试玩家所发游戏截图的数量
>> testx.shape=(8,1)
>>> model.apply([50]) # 应用刚才建立的决策树分类模型, 对某个测试样本进行分类
False
>> model.apply_many(testx) # 应用刚才建立的决策树分类模型, 对成批测试样本进行分类
[False, False, False, True, False, False, True, True]
```

观察最后两行程序代码的返回结果, 倒数第二行的程序代码对 50 进行了测试, 通过决策树模型判别, 发游戏截图 50 次的用户为非热心玩家, 倒数第一行程序代码对一批测试样本进行判断, 其中发游戏截图为 90、199、200 次的玩家值返回为 True, 表示这些用户为热心玩家。

接下来, 看一个较复杂的例子。假设某公司针对其赊账客户进行分类, 通过随机抽取 7 位赊账客户进行评估, 之后将其分成可容忍客户和不容忍客户, 并以这些客户为样本数据, 尝试对更多的赊账客户进行决策树分类。7 位赊账客户的数据如表 8-6 所示。

表 8-6 赊账客户的数据

到目前为止仍欠款	欠款笔数	到目前为止累计拖欠天数	是否容忍
5600	4	50	Yes
58 000	6	120	NO
1300	1	8	YES
2900	3	12	YES
800	2	5	YES
35 000	4	190	NO
9800	2	10	NO

应用 Python 对以上数据进行决策树算法分类, 代码如下所示:

```
>>
x=np.array([[5600,58000,1300,2900,800,35000,9800],[4,6,1,3,2,4,2],[50,120,8,12,5,190,10]])# 建立样本数据
>> features=x.T
>> features
array([[ 5600,    4,   50],
       [58000,    6,  120],
       [ 1300,    1,    8],
       [ 2900,    3,   12],
       [   800,    2,    5],
       [35000,    4,  190],
       [ 9800,    2,   10]])
# 建立决策树模型
>> learner=milk.supervised.tree_learner()
>> model=learner.train(features, labels)
# 对到目前为仍欠款 1200, 欠款笔数为 3, 到目前为为止累计拖欠 13 天的客户进行判断, 确定其是否属于可容忍客户, 经决策树模型分析, 可以容忍。
>> model.apply([1200,3,13])
True
# 对到目前为仍欠款 12000, 欠款笔数为 13, 到目前为为止累计拖欠 103 天的客户进行判断, 确定其是否属于可容忍客户, 经决策树模型分析, 不能容忍。
>> model.apply([12000,13,103])
False
```

8.9.3 AdaBoost

AdaBoost 分类算法是一种迭代分类算法，它会在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率为止。每一个训练样本都被赋予一个权重，表明它被某个分类器选入训练集的概率。如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它被选中的概率就会被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到了提高。通过这种方式，AdaBoost 方法能聚焦于那些较难分（更富信息）的样本上。该算法的核心思想如下：

最初令每个样本的权重都相等，对于第  $k$  次迭代操作，我们就根据这些权重来选取样本点，进而训练分类器  $C_k$ ；然后就根据这个分类器，来提高被它分错的样本的权重，并降低被正确分类的样本的权重；最后，更新过权重的样本集被用于训练下一个分类器  $C_{k+1}$ 。整个训练过程如此迭代地进行下去。

AdaBoost 算法的具体过程如下：

- 1) 先通过对  $N$  个训练样本的学习得到第一个弱分类器。
- 2) 将分错的样本和其他的新数据一起构成一个新的  $N$  个的训练样本，通过对这个样本的学习得到第二个弱分类器。
- 3) 将步骤 1 和 2 中分错了的样本加上其他的新样本构成另一个新的  $N$  个的训练样本，通过对这个样本的学习得到第三个弱分类器。
- 4) 如此迭代，最终经过提升得到强分类器。

继续以上一节的赊账客户为例进行说明，假设该公司随机抽取 7 位赊账客户进行评估，之后按信用等级对其进行分类（共分为 3 级，1 级信用等级最差，3 级信用等级最好）并以这些客户为样本数据，尝试对更多的赊账客户进行 AdaBoost 算法分类。7 位赊账客户的数据如表 8-7 所示。

表 8-7 赊账客户的数据

到目前为止仍欠款	欠款笔数	到目前为止累计拖欠天数	信用等级
5600	4	50	2
58 000	6	120	1
1300	1	8	3
2900	3	12	2
800	2	5	3
35 000	4	190	1
9800	2	10	2

首先，建立分类器，Python 代码如下：

```
# 导入相关库
>> import milk.supervised.tree
>> import milk.supervised.adaboost
>> import milk.supervised.multi
# 建立分类器
>> weak = milk.supervised.tree.stump_learner()
```



```
>> learner = milk.supervised.adaboost.boost_learner(weak)
>> learner = milk.supervised.multi.one_against_one(learner)
# 样本数据及样本分类
>>
x=np.array([[5600,58000,1300,2900,800,35000,9800],[4,6,1,3,2,4,2],[50,120,8,
12,5,190,10]])
>> features=x.T
>> labels=np.array([2,1,3,2,3,1,2])
>> learner.train(features,labels)
```

然后，对测试样本进行预测，返回信用等级，测试数据的第一个元素是到目前为止仍欠款，第二个元素是欠款笔数，第三个元素是到目前为止累计拖欠天数。

```
>>> model.apply([1000,3,10])
3
>>> model.apply([10000,3,10])
2
>>> model.apply([10000,5,10])
2
>>> model.apply([10000,5,100])
2
>>> model.apply([50000,5,100])
1
>>> model.apply([50000,5,10])
1
>>> model.apply([5000,15,100])
2
>>> model.apply([5000,15,10])
2
>>> model.apply([28000,15,100])
2
>>> model.apply([38000,15,10])
1
```

分析以上结果，以 `model.apply([38000,15,10])` 为例，返回 1 表示目前为止仍欠款 38 000，欠款笔数 15 次，目前为止累计拖欠 10 天的信用等级为 1 级，信用等级最差。

#### 8.9.4 竞争型神经网络

神经生物学的研究表明：生物视网膜有许多特定的细胞，对特定的图形输入模式比较敏感，并使得大脑皮层中的特定细胞产生较大的兴奋，而与其相邻的神经细胞的兴奋程度则被抑制。竞争型神经网络对此进行了模拟，对于某一个输入模式，通过竞争在输出层中只激活一个相应的输出神经元；若有许多输入模式，则在输出层中激活许多个神经元，从而形成一个反映输入数据的“特征图形”，可实现对输入模式自动进行分类。

竞争型神经网络一般是由输入层（模拟视网膜神经元）和竞争层（模拟大脑皮层神经元，也叫输出层）构成的两层网络，两层中的各神经元之间实现双向全连接，而且网络中没有隐含层，有时竞争层各神经元之间还存在横向连接，对于某一输入模式，在竞争型神经网络中，和该模式最相近的学习输入模式相对应的竞争层神经元将有最大的输出值，即以竞争层获胜神经元来表示分类结果。竞争型神经网络结构如图 8-56 所示。



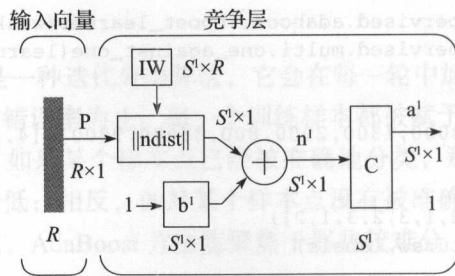


图 8-56 竞争型神经网络

程序 8-25.py 演示了竞争型神经网络：

```
# -*- coding: utf-8 -*-
#8-25.py

import numpy as np
import neurolab as nl
import numpy.random as rand

# 每一类的中心
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.3]])
# 以每类的中心为基础，产生随机点。
rand_norm = 0.05 * rand.randn(100, 3, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 3, 2)
rand.shuffle(inp)

# Create net with 2 inputs and 3 neurons
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 3)
# 训练该神经网络
# train with rule: Conscience Winner Take All algorithm (CWTA)
error = net.train(inp, epochs=200, show=20)

# Plot results:
import pylab as pl
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default MAE)')
w = net.layers[0].np['w']

pl.subplot(212)
pl.plot(inp[:,0], inp[:,1], '.', \
        centr[:,0], centr[:,1], 'yv', \
        w[:,0], w[:,1], 'p')
pl.legend(['train samples', 'real centers', 'train centers'],loc=2)
pl.show()
```

程序 8-25.py 创建了 3 个神经元，接受 2 个输入，输出为 3 类。运行程序 8-25.py，输出结果如图 8-57 所示。

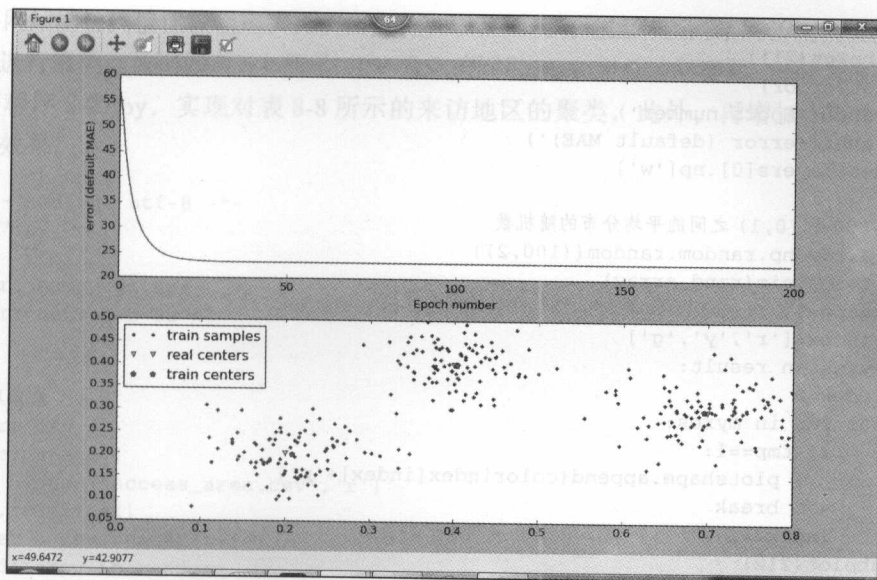


图 8-57 竞争型神经网络分类

观察图 8-57，程序将训练用样本点聚集为 3 个类别，其中，train samples 表示训练样本，real centers 表示真实中心，train centers 表示训练中心。

下面在程序 8-25.py 的基础上加入测试用的平均随机分布点，检查其分类效果，如程序 8-26.py 所示：

```
# -*- coding: utf-8 -*-
#8-26.py

import numpy as np
import neurolab as nl
import numpy.random as rand

# 每一类的中心
centr = np.array([[0.2, 0.2], [0.4, 0.4], [0.7, 0.8]])
# 以每类的中心为基础，产生随机点。
rand_norm = 0.05 * rand.randn(100, 3, 2)
inp = np.array([centr + r for r in rand_norm])
inp.shape = (100 * 3, 2)
rand.shuffle(inp)

# Create net with 2 inputs and 3 neurons
net = nl.net.newc([[0.0, 1.0], [0.0, 1.0]], 3)
# 训练该神经网络
# train with rule: Conscience Winner Take All algorithm (CWTA)
error = net.train(inp, epochs=200, show=20)

# Plot results:
import pylab as pl
```

```
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default MAE)')
w = net.layers[0].np['w']

# 生成 100 个 [0,1) 之间的平均分布的随机数
rand_array=np.random.random((100,2))
result=net.sim(rand_array)
plotshape=[]
colorindex=['r','y','g']
for myres in result:
    index=0
    for tmp in myres:
        if tmp==1:
            plotshape.append(colorindex[index]+'p')
            break
        index+=1
pl.subplot(212)
pl.plot(inp[:,0], inp[:,1], '.')
for i in range(len(result)-1):
    pl.plot(rand_array[i,0],rand_array[i, 1],plotshape[i])
pl.show()
```

程序 8-26.py 生成了 100 个 [0,1) 之间的平均分布的随机数，然后由训练完成的神经网络进行分类。运行程序 8-26.py，效果如图 8-58 所示，训练样本点为蓝色圆形，测试样本点为多边形，测试样本点的不同色彩代表了被分为不同的类。此外，图 8-58 上方的图表示误差，从误差曲线来看，神经网络训练过程顺利，呈平滑下降趋势。

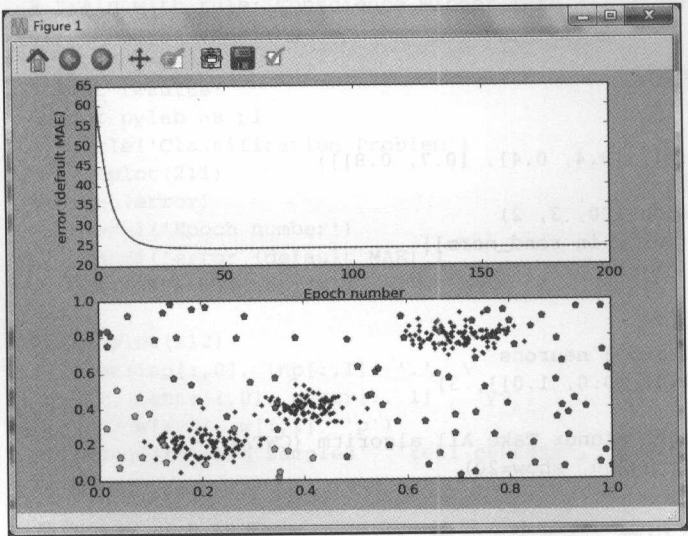


图 8-58 测试点聚类

表 8-8 网站的访问量情况

area	PV	UV
1	106	37
2	186	54
3	128	39
4	89	34
5	23	18
6	63	27
7	62	30
8	62	36
9	120	38
10	110	47
11	94	45
12	100	37
13	67	27
14	120	31
...	...	...

设某网站的访问量比较稳定,无突出变化,以某周所有地区对该网站的访问情况为例,对来访地区进行聚类,数据如表 8-8 所示(PV 为总访问量,UV 为总访问客数,area 为地区编号)。

编写程序 8-27.py,实现对表 8-8 所示的来访地区的聚类,此外,再增加 100 个随机点,检测聚类效果。

```
# -*- coding: utf-8 -*-
#8-27.py

import numpy as np
import neurolab as nl

# 读取数据
import csv
datacluster=[]
file = open("access_area.csv",'r')
file.readline()
reader = csv.reader(file)
for datarow in reader:
    datacluster.append([float(datarow[1]),float(datarow[2])])

accessdata=np.array(datacluster)

# Create net with 2 inputs and 7 neurons, 分为 7 类
net = nl.net.newc([[0.0, max(accessdata[:,0])],[0.0, max(accessdata[:,1])]],7)
# 训练该神经网络
# train with rule: Conscience Winner Take All algorithm (CWTA)
error = net.train(accessdata, epochs=200, show=20)

# Plot results:
import pylab as pl
pl.title('Classification Problem')
pl.subplot(211)
pl.plot(error)
pl.xlabel('Epoch number')
pl.ylabel('error (default MAE)')
w = net.layers[0].np['w']

# 生成 100 个平均分布的随机数
rand_array=np.random.random((100,2))*np.array([max(accessdata[:,0]),max(accessdata[:,1])])
result=net.sim(rand_array)
plotshape=[]
colorindex=['r','y','g','c','m','k','b']
for myres in result:
    index=0
    for tmp in myres:
        if tmp==1:
            plotshape.append(colorindex[index]+'p')
            break
```

```

index+=1
w = net.layers[0].np['w']
pl.subplot(212)
pl.xlabel('PV')
pl.ylabel('UV')
pl.plot(accessdata[:,0], accessdata[:,1], '.')

for i in range(len(result)-1):
    pl.plot(rand_array[i,0],rand_array[i, 1],plotshape[i])
for i in xrange(7):
    pl.plot(w[i,0], w[i,1],colorindex[i]+'v')

pl.show()

Epoch: 20; Error: 786.921489826;
Epoch: 40; Error: 784.125494497;
Epoch: 60; Error: 783.165674735;
Epoch: 80; Error: 782.677433443;
Epoch: 100; Error: 782.381170541;
Epoch: 120; Error: 782.182073001;
Epoch: 140; Error: 782.039000206;
Epoch: 160; Error: 781.931187336;
Epoch: 180; Error: 781.84701229;
Epoch: 200; Error: 781.77945951;
The maximum number of train epochs is reached
>>>

```

运行程序 8-27.py, 将样本聚成 7 类, 同时将测试点较好地完成分类。效果如图 8-59 所示。

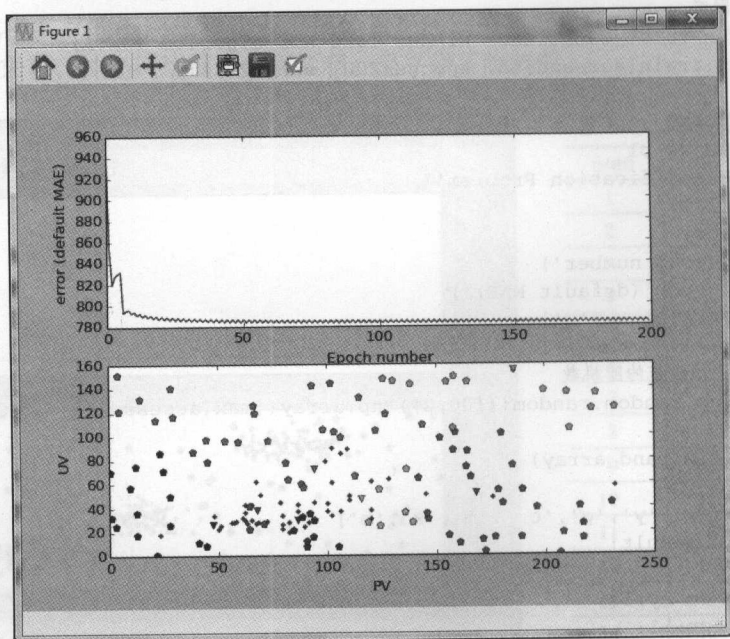


图 8-59 地区聚类



观察图 8-59, 不同色彩的多边形代表了测试点被分成了 7 类, 此外, 程序还绘制了样本点及倒三角代表的类别中心点, 总体来看, 效果还不错。

### 8.9.5 Hamming 神经网络

图 8-60 是最典型的 Hamming 神经网络, 第一层上面的方框是输入数据, 每个输入数据都会与神经元相连。每个神经元的一组连接权值存储的是一个识别对象的模板, 神经元的作用就是计算输入数据与其连接权值所存模板的匹配距离, 匹配程度由神经元输出, 匹配度越高, 输出值越大。

假设销售某虚拟物品时, 向客户展示了一些相关的畅销虚拟物品, 以表 8-9 所示的数据为例, 将消费兴趣类似的客户分类。

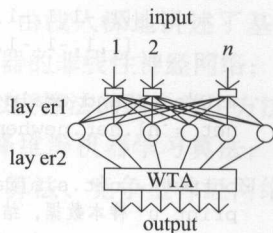


图 8-60 Hamming 神经网络

表 8-9 点击数据

A1	A2	A3	A4	B1	B2	B3	B4	...
1	-1	1	1	-1	1	1	-1	...
-1	-1	-1	1	1	-1	1	1	...
-1	-1	-1	-1	-1	-1	-1	1	...
1	-1	1	-1	1	1	1	1	...
1	1	-1	-1	1	-1	-1	-1	...

表 8-9 中, A1 表示 A 类虚拟物品的 1 号商品, B2 表示 B 类虚拟物品的 2 号商品, 以此类推。每一行代表一类, 每列数据代表该类客户浏览的虚拟商品, 如果该虚拟商品点击量较多, 则为 1, 否则为 -1。

编写 Python 代码进行聚类 (sales4.csv 在本书源代码包中), 如程序 8-28.py 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
#8-28.py
import numpy as np
import neurolab as nl

# 读取数据
import csv
datatarget=[]
file = open("sales4.csv",'r')
file.readline()
reader = csv.reader(file)
ii=0
for datarow in reader:
    datatarget.append([])
    for data in datarow:
        datatarget[ii].append(int(data))
```

```

ii=ii+1

datatarget=np.array(datatarget)

input = [[1,1,1,1,-1,1,1,-1,-1,-1,-1],
         [1,-1,1,-1,1,-1,1,1,-1,-1,-1],
         [1,1,-1,-1,1,-1,-1,-1,-1,1,1,-1]]

# Create and train network
net = nl.net.newhem(datatarget)

output = net.sim(datatarget)
print u" 样本数据, 结果必须为 [0, 1, 2, 3, 4]"
print np.argmax(output, axis=0)

output = net.sim(input)
print u" 测试数据的神经网络最终输出 "
print output
print u" 测试数据的分类结果如下: "
ii=0
for test in output:
    print input[ii],
    print u" 分类如下: "
    print np.argmax(test)
    ii+=1

```

程序 8-28.py 首先读取样本 sales4.csv 文件; 然后将样本数据送入 Hamming 神经网络中进行训练, 匹配程度由神经元输出, 匹配度越高, 输出值越大, 通常来说最大值对应的位置即最终分类; 最后, 用测试数据进行验证。执行程序 8-28.py, 结果如下。

样本数据的结果必须为 [0,1,2,3,4]。

```
[0 1 2 3 4]
```

测试数据神经网络的最终输出如下:

```

[[ 0.52608 0.      0.      0.      0.      ]
 [ 0.      0.      0.      0.464  0.      ]
 [ 0.      0.      0.      0.      0.4992 ]]

```

来看看测试数据的分类结果。

[1,1,1,1,-1,1,1,-1,-1,-1,-1] 分类如下:

```
0
```

[1,-1,1,-1,1,-1,1,1,-1,-1,1] 分类如下:

```
3
```

[1,1,-1,-1,1,-1,-1,-1,1,1,-1] 分类如下:

```
4
```

观察上述结果,三组测试数据均分类准确,效果不错。

## 8.10 小结

机器学习算法是机器学习的灵魂。本章首先介绍神经网络,由浅入深地讲述了基于 Rosenblatt 感知器的线性神经网络、反向传播算法及基于多层感知器的非线性神经网络;然后介绍了平均值与方差、贝叶斯分类等统计算法;接着阐述了相似度算法的两个常用方法:欧氏距离和余弦相似度;再然后讲解了 SVM、回归算法、PCA 降维等机器学习算法;最后讲解了关联规则算法、常用聚类算法、决策树算法、AdaBoost 算法、竞争型神经网络、Hamming 神经网络等算法知识。

在讲述算法的同时,笔者理论联系实际,讲解了如何用 Python 及相关库、R 语言来实践机器学习算法。此外,为帮助读者更好地理解算法,在解说每一节的算法之前,均介绍了算法涉及的数学基础知识。

每一种机器学习算法不一定能完成所有机器学习的任务。因此,应在实践中应用这些算法,观察算法的实际效果,以求选择最合适的算法。

此外,本章介绍的机器学习相关库的官方文档的网址如下:

□ mlp: <http://sourceforge.net/projects/mlpy/files/mlpy%203.5.0/mlpy.pdf/download>

□ neurolab: <https://pythonhosted.org/neurolab/index.html>

## 思考题

(1) 编写 Python 代码实现一个 Rosenblatt 感知器,随机产生一组  $X$  和  $Y$  的样本值,这些样本值分别属于以下两类函数:

$$5x-3=y \text{ 为第 1 类}$$

$$2x+6=y \text{ 为第 2 类}$$

然后用随机生成的测试数据进行分类,并绘制出分类线。

(2) 编写 Python 代码实现多层感知器,随机产生一组  $X$  和  $Y$  的样本值,用随机数据对训练后的网络进行测试绘制散点图和误差曲线,计算  $X$  和  $Y$  的方差和平均值,分析其分布趋势。

(3) 编写 Python 代码实现多层感知器,绘制样本散点图和误差曲线。随机产生一组  $X$  样本值,并将  $Y$  的样本值函数定义为:

$$Y=\sin(x)*0.7$$

(4) 编写 Python 代码实现 SVM 算法,随机产生一组  $X$  和  $Y$  的样本值,这些样本值分别属于以下两类函数:

$$y=x^a+b \quad (a \leq 3, \text{abs}(b) < 20) \text{ 为第一类}$$

$$y=x^a+b \quad (a \geq 5, \text{abs}(b) < 10) \text{ 为第二类}$$

然后用随机生成的测试数据进行分类,并绘制出散点图。

(5) 选择一组数据，用 Python 实现下面的回归方程：

$$y=b1*x2+b2*x1^2+b3*x1*x2$$

(6) 下载本书的 sales4.csv 文件，加入更多类别的样本数据，并生成一些随机测试数据，进行 Hamming 神经网络分类，并测试分类效果。

(7) 生成 0 ~ 9 共 9 个数字的印刷体样本，并生成一些测试样本数据，进行 Hamming 神经网络分类，检查分类效果。

提示：可将数字的像素点组成矩阵，然后展开成特征向量。比如：8 可表示为以下由像素点组成的矩阵（1 表示有像素点，-1 表示无像素点）：

1	1	1
1	-1	1
1	1	1
1	-1	1
1	1	1

将 8 的像素点矩阵由左到右、由上到下依次展开，生成特征值如下：[1,1,1,1,-1,1,1,1,1,-1,1,1,1,1]。

(8) 下表是某些顾客对 A ~ D 四类商品的季度消费金额，对这些顾客运用本章讲解的各种聚类方法进行聚类。

ID	A	B	C	D
1	190	300	10	20
2	20	400	201	20
3	50	60	500	82
4	700	1200	300	320
5	1800	1500	300	10
6	200	3000	90	120
7	100	40	1200	400

## 数据拟合案例

在科学和工程问题上可通过采样、实验等方法获得若干离散的数据,分析这些数据能得到一个连续的函数(曲线)或者更加密集的离散方程,且这个方程与已知数据相吻合,这个过程叫做数据拟合。数据拟合实质是针对一组未知规律的数据建立数学方程,通过数学方法建立一个函数映射方式。

前面几章涉及了很多回归分析算法,都是先根据样本建立模型,然后分析回归效果,最终目的是弄清楚两个或两个以上变量之间的因果关系,因此,这些算法都是对数据的拟合。

### 9.1 数据拟合

分析一个自变量和一个因变量之间的关系,可通过图像分析法与神经网络拟合法建立数学模型进行映射。

#### 9.1.1 图像分析法

顾名思义,图像分析法就是将样本数据中的自变量 $X$ 和因变量 $Y$ 的值映射为平面直角坐标系中的点,其 $X$ 轴坐标和 $Y$ 轴坐标的值分别为自变量和因变量的值,这些点共同形成一个散点图。通过分析散点图中这些点的几何分布趋势,对照常见函数图像,选择最接近的、最适合的数学函数作为拟合方程。

既然是图像分析法,那么就需要对常见的数学函数及其图像有一个直观的认识。因此,这里先介绍常见的数学函数。

##### 1. 幂函数

幂函数是形如 $f(x)=x^a$ 的函数,函数以底数为自变量,幂为因变量,指数 $a$ 为常量, $a$ 可



以是自然数、有理数、任意实数或复数。下面是几个经典的幂函数。

□ 指数为 1、2、1/2 的幂函数，它们分别为  $y=x$ 、 $y=x^2$ 、 $y=x^{1/2}$ ，图像如图 9-1 所示。

□ 指数为 3、1/3 的幂函数，它们分别为  $y=x^3$ 、 $y=x^{1/3}$ ，图像如图 9-2 所示。

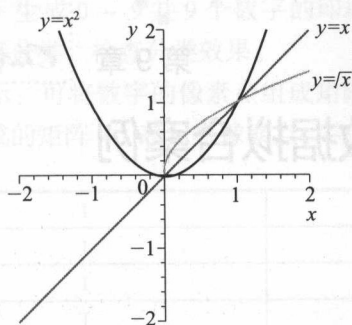


图 9-1 指数为 1、2、1/2 的幂函数图像

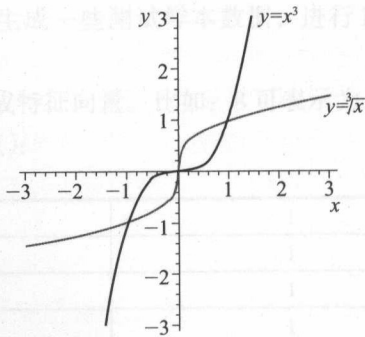


图 9-2 指数为 3、1/3 的幂函数图像

□ 指数为 -1 的幂函数，函数为  $y=x^{-1}$ ，图像如图 9-3 所示。

## 2. 一次函数

一次函数又称线性函数，是指拥有一个变量的一阶多项式函数。一次函数可以表达为以下斜截式的格式：

$$f(x)=kx+b$$

其中， $k$  是斜率， $b$  是截距。

一次函数在二维坐标系中表现为一条直线，以自变量为  $X$  轴，以因变量为  $Y$  轴。如： $y=2x+1$  和  $y=-x+1$  属于一次函数，它们的图像如图 9-4 所示。

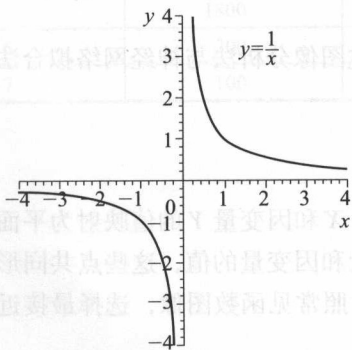


图 9-3 指数为 -1 的幂函数图像

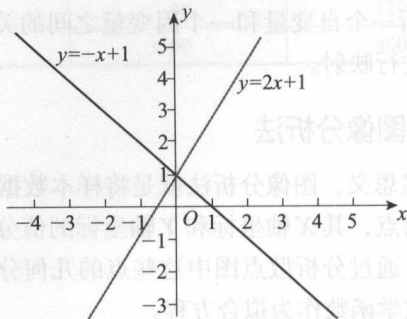


图 9-4 一次函数图像

## 3. 一元二次函数

如果  $y=ax^2+bx+c$  ( $a$ 、 $b$ 、 $c$  为常数， $a$  不为 0)，则称  $y$  为  $x$  的一元二次函数，其中， $a$  称

为二次项系数,  $b$  为一次项系数,  $c$  为常数项。

一元二次函数是抛物线, 是轴对称图形, 它的对称轴为直线  $x = -\frac{b}{2a}$ , 它的顶点坐标为  $\left(-\frac{b}{2a}, \frac{4ac-b^2}{4a}\right)$ 。此外, 一元二次函数还可写成交点式:  $y=a(x-x_1)(x-x_2)$ , 交点式仅限于与  $x$  轴有交点的抛物线, 与  $x$  轴的交点坐标是  $(x_1, 0)$  和  $(x_2, 0)$ 。

如图 9-5 所示为一元二次函数  $y=2x^2+x+1$  和  $y=-2x^2+x+2$  的图像。从图像上观察, 一元二次函数是一个开口向上或开口向下的抛物线, 二次项系数  $a$  决定抛物线的开口方向和大小, 当  $a>0$  时, 抛物线向上开口; 当  $a<0$  时, 抛物线向下开口。 $|a|$  越大, 则抛物线的开口越小。

#### 4. 指数函数

指数函数  $y=e^x$  是数学中重要的函数, 这里的  $e$  是数学常数, 就是自然对数的底数, 近似等于 2.718281828。 $y=e^x$  的图像总在  $x$  轴之上并从左向右递增, 它接近  $x$  轴但不会与  $x$  轴相交。

此外, 还可定义更一般的指数函数  $y=a^x$ , 对于  $a>0$  和实数  $x$ , 该函数可称为底数为  $a$  的指数函数。 $a^x$  可如下变换为以  $e$  为底的指数函数。

$$a^x = (e^{\ln a})^x = e^{x \ln a}$$

观察图 9-7 所示的指数函数  $y=a^x$ , 当  $a>1$  时, 从左到右函数是递增的, 否则函数是递减的。指数函数遵循以下运算规律:

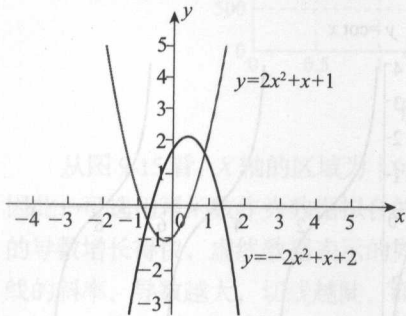


图 9-5 二次函数图像

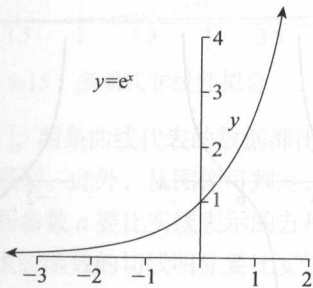


图 9-6 指数函数  $y=e^x$

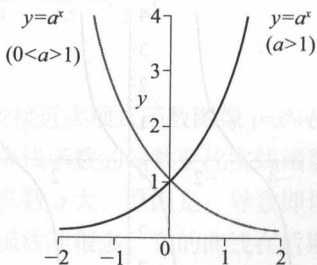


图 9-7 指数函数  $y=a^x$

$$a^0 = 1$$

$$a^1 = a$$

$$a^{x+y} = a^x a^y$$

$$a^{xy} = (a^x)^y$$

$$\frac{1}{a^x} = \left(\frac{1}{a}\right)^x = a^{-x}$$

$$a^x b^x = (ab)^x$$

#### 5. 对数函数

相对于底数  $a$  数  $x$  的对数是  $a^y$  的指数  $y$ , 使得  $x=a^y$ , 相对于底数  $a$  的数  $x$  的对数通常记

为 $y=\log_a x$ ，在工程计算中常用 $e$ 、 $10$ 和 $2$ 来做底数，如表9-1所示。

观察图9-8所示的对数函数图像，可看出：当 $a>1$ 时，函数是递增的，否则，函数是递减的。自然对数的底 $e$ 大于1，它的函数图像是递增的，如图9-9所示。

表 9-1 常用对数函数表示

底数	名称	表示
2	二进制对数	$\lg x$
10	常用对数	$\lg x$
$e$	自然对数	$\ln x$

6. 三角函数

三角函数中使用得最多的是正弦（如图9-10所示）、余弦（如图9-10所示）、正切（如图9-11所示）、余切（如图9-12所示）。

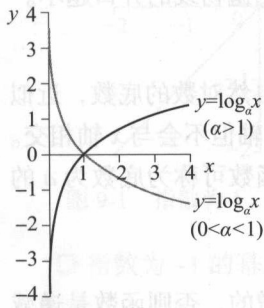


图 9-8  $y=\log_a x$

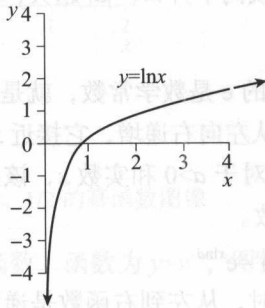


图 9-9 自然对数

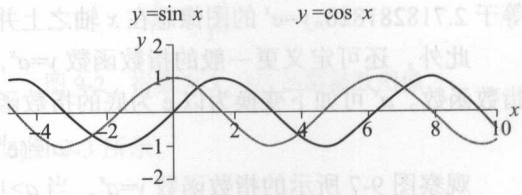


图 9-10 正弦函数与余弦函数

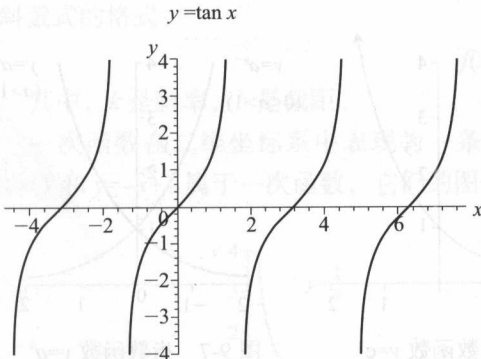


图 9-11 正切函数

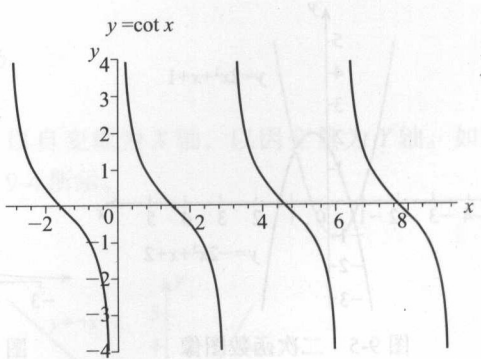


图 9-12 余切函数

7. 多项式函数

多项式的函数图像呈现连续曲线，以多项式 $y = \frac{x^3}{4} + \frac{3x^3}{4} - \frac{3x}{2} - 2$ 和 $y=x^2-x-2$ 为例，其函数图像如图9-13和图9-14所示。

仔细观察前面列举的函数图像，可发现常见的函数分为两种：线性函数与非线性函数。其中，线性函数是变量与自变量成一次方的函数关系，在函数图上呈现一条直线；而非线性函数是指两个变量间的关系不成简单比例，函数图像呈现为曲线。

线性拟合是以一条直线来拟合自变量与因变量的关系，而非线性拟合用连续曲线来拟合自变量与因变量之间的关系。下面以几个具体实例说明图像分析法在线性与非线性拟合中的应用。

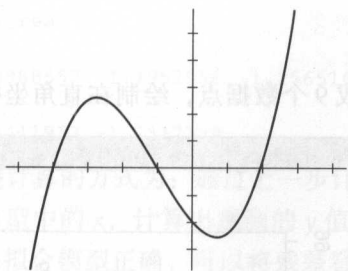


图 9-13  $y = \frac{x^3}{4} + \frac{3x^3}{4} - \frac{3x}{2} - 2$  的函数图像

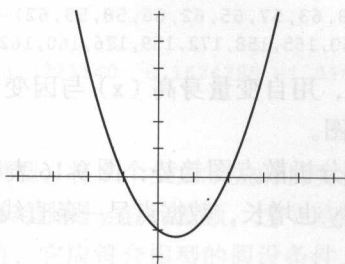


图 9-14  $y = x^2 - x - 2$  的函数图像

1) 多项式是非线性函数, 如图 9-15 所示的图像为两类样本数据生成的散点图, 其中虚线和实线各代表一类。

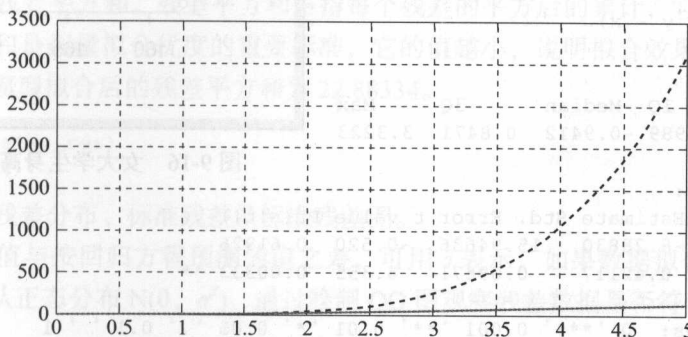


图 9-15 多项式非线性拟合

从图 9-15 看,  $X$  轴的区域为  $[0, 5]$ , 两条曲线代表的数据都比较接近多项式函数图像  $y = x^a + b$ , 因此, 可选用幂函数作为数据拟合的模型。此外, 从图像可判定, 虚线函数的导数要比实线函数的导数增长得快, 虚线数据表示的方程参数  $a$  要比实线表示的方程参数  $a$  大, 原因是: 导数即切线的斜率, 导数越大, 切线越陡, 而虚线函数的切线明显要比实线函数陡很多, 它的曲线在后期更加上扬。事实上, 图 9-15 所代表的两类数据的模型分别为: 虚线是  $y = x^5 + 6$ , 实线是  $y = x^3 + 6$ 。

2) 大学生身高与体重线性拟合。从某大学中随机选取 9 名女大学生, 其身高和体重数据如表 9-2 所示。下面分析这些数据并建立女大学生身高与体重模型, 从而指导女大学生通过简易的公式查询自己的体重是否标准。

表 9-2 女大学生身高与体重数据

编号	1	2	3	4	5	6	7	8	9
身高/cm	160	165	158	172	159	176	160	162	171
体重/kg	58	63	57	65	62	66	58	59	62

首先, 使用 R 语言输入数据 (本例只有 9 个数据, 可手工录入。当数据量很大时, 可使用第 6 章介绍的 `read.table` 函数读取数据)。

```
>c(58,63,57,65,62,66,58,59,62)->y
>c(160,165,158,172,159,176,160,162,171)->x
```

接着,用自变量身高(x)与因变量体重(y)组成9个数据点,绘制在直角坐标系中,生成散点图。

观察分析散点图趋势,图9-16表明,随着x的增长,y也增长,数据点呈一条直线分布(线性拟合)。

下面以直线模型对数据进行拟合,可通过R语言进行拟合分析。

```
> lm(y~x)->xy
> summary(xy)
Call:
lm(formula = y ~ x)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-1.7317 -1.0989 -0.9412  0.8471  3.3223
```

Coefficients:

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -8.28830    15.94636   -0.520  0.61926
x             0.42117     0.09671    4.355  0.00333 **
```

---

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1.808 on 7 degrees of freedom
```

```
Multiple R-squared:  0.7304,    Adjusted R-squared:  0.6919
```

```
F-statistic: 18.97 on 1 and 7 DF,  p-value: 0.003334
```

通过对上述直线方程模型的分析,可初步得出以下结论:

□ 直线方程即线性方程,可写成 $y=ax+b$ 的方式, $a$ 即系数(Coefficients项中的 $x$ )为0.42117, $b$ 即截距(Coefficients项中的Intercept)为-8.28830,线性拟合模型为 $y=0.42117x-8.28830$ 。

□ Coefficients栏的 $x$ 行尾的星号为2个。星号的含义表示线性关系是否显著: \* 的数量是0~3, \* 的数量越多则线性关系越显著。在本例中,自变量身高与因变量体重的线性关系并不是非常显著,否则星号应为3个。

下面绘制回归线,图9-17所示。观察各数据点在回归线周围的分布情况,目测拟合效果。

```
> plot(x,y)
> abline( lm(y~x))
```

从图9-17来看,数据分布在回归线周围,但某些数据点的残差较大(比如155到160身高段)。总体来说,数据拟合效果较好,没有出现数据点分布趋势严重偏离回归线。

通过R语言的residuals函数分析残差。

```
> residuals(xy)->xy_res
```

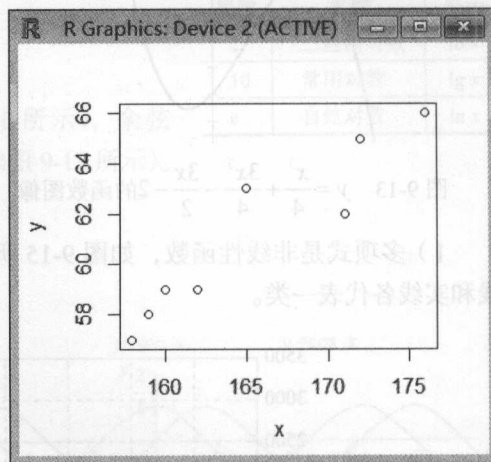


图9-16 女大学生身高体重散点图



```
> xy_res
      1      2      3      4      5      6      7
-1.0988557  1.7952956 -1.2565162  0.8471074  3.3223140  0.1624285 -1.0988557
      8      9
-0.9411952 -1.7317228
```

残差计算的方式为：通过上一步计算得到的线性模型，将9个女大学生的自变量身高带入线性模型中的  $x$ ，计算出观测的  $y$  值后，计算实际值与预测  $y$  值的差额，这个差额就是残差。如果拟合模型正确，可以将残差看作误差的观测值，它应符合模型的假设条件，且具有误差的一些性质。

上述代码根据线性模型计算得出残差对象 `xy_res`，9个元素分别为9个女大学生样本数据的残差，其中残差最大的数据为第5个女大学生，即3.3223140，最小为0.1624285，是第6个女大学生。

然后，计算残差平方和。残差平方和是指每个残差的平方后的累计，它表示随机误差的效应。残差平方和是衡量拟合优度的重要标准，它的值越小，说明拟合效果越好。通过下面代码计算，线性模型拟合后的残差平方和为22.88334。

```
> sum(xy_res*xy_res)
[1] 22.88334
```

最后，分析残差分布、标准残差及标准残差图。

残差为测定值与按回归方程预测的值之差，可用  $\delta$  表示，如果数据拟合模型效果较好，那么残差  $\delta$  应遵从正态分布  $N(0, \sigma^2)$ ，通过绘制 QQ 图观察残差数据是否符合正态分布。

```
> qqnorm(xy_res)
> qqline(xy_res)
```

观察残差分布 QQ 图，如图 9-18 所示。可看出：残差  $\delta$  分布接近正态分布，数据点基本在直线附近，回归效果较理想，但仍不是非常理想。

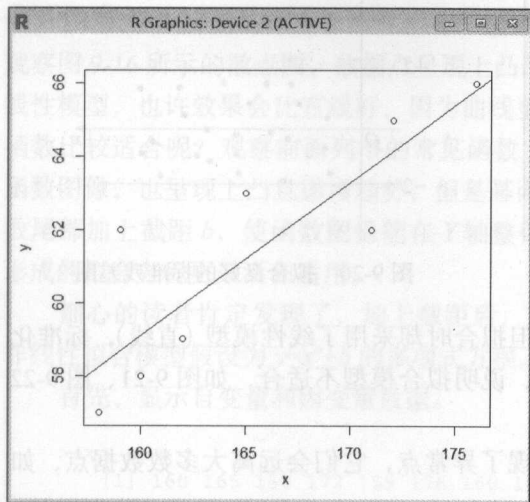


图 9-17 女大学生身高体重数据的回归线

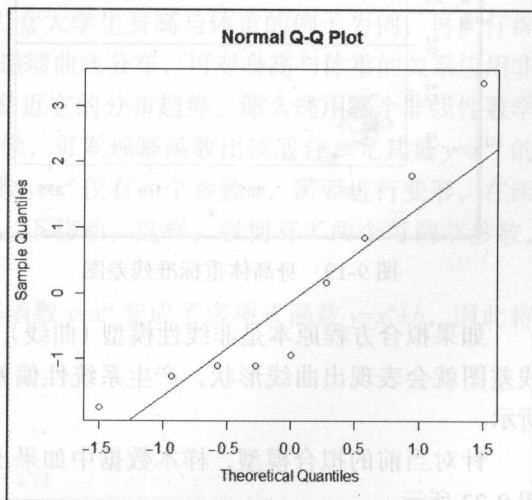


图 9-18 残差分布 QQ 图

标准残差就是(残差 $\delta$ -残差的均值)/残差的标准差,用 $\delta^*$ 表示。下面通过 R 语言的 `rstandard` 函数计算身高体重拟合模型的标准残差,同时绘制标准残差图,图 9-19 所示。

```
> rstandard(xy)->std_xy_res
> std_xy_res
      1      2      3      4      5      6      7
-0.6696927 1.0532610 -0.7984996 0.5447647 2.0629420 0.1235619 -0.6696927
      8      9
-0.5591209 -1.0857787
> plot(x,std_xy_res)
> abline(h=0)
```

标准残差图是以拟合模型的自变量为横坐标,以标准残差为纵坐标,将每一个自变量的标准残差描述在该平面坐标上,形成图形,实验点的标准残差落在残差图的 $(-2, 2)$ 区间以外的概率 $\leq 0.05$ ,若某一实验点的标准化残差落在 $(-2, 2)$ 区间以外,可在 95% 置信度将其判为异常实验点。置信度也称置信水平,抽样不可能抽取全部总体进行分析,由于样本的随机性,通过这些样本对总体参数作出估计时,结论总是不确定的,因此引入置信度,用概率来陈述总体参数值落在样本统计值某一区内的可能性。

当描绘标准残差的点围绕标准残差等于 0 的直线上下完全随机地分布,绝大多数点落在 $(-2, +2)$ 的水平带状区间之中,且不带有任何系统趋势时,则说明拟合模型对原观测值(即样本)的拟合情况良好,如图 9-20 所示。

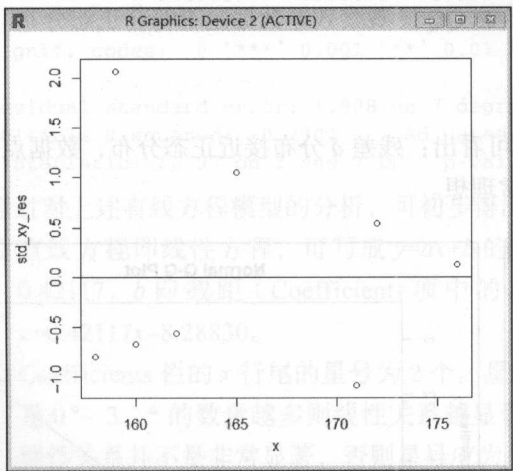


图 9-19 身高体重标准残差图

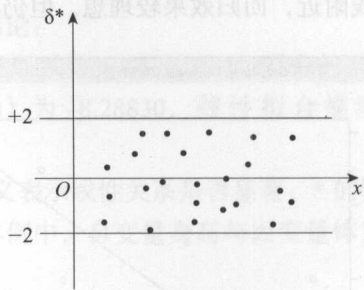


图 9-20 拟合良好的标准残差图

如果拟合方程原本是非线性模型(曲线),但拟合时却采用了线性模型(直线),标准化残差图就会表现出曲线形状,产生系统性偏差,说明拟合模型不适合,如图 9-21、图 9-22 所示。

针对当前的拟合模型,样本数据中如果出现了异常点,它们会远离大多数数据点,如图 9-23 所示。

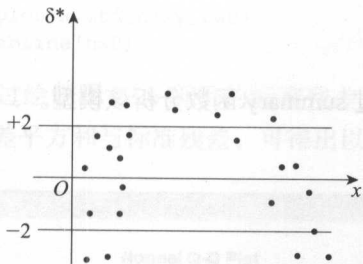


图 9-21 出现系统性偏差的标准残差图①

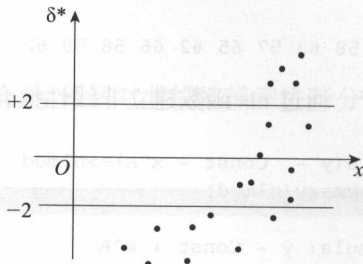


图 9-22 出现系统性偏差的标准残差图②

此外, 如果拟合不充分, 很多点会落在  $(-2, +2)$  的水平带状区间之外, 如图 9-24 所示。

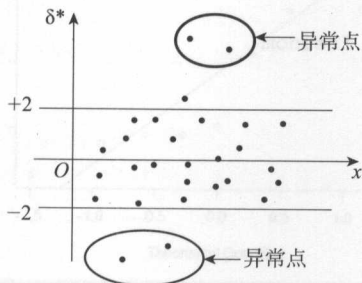


图 9-23 异常点

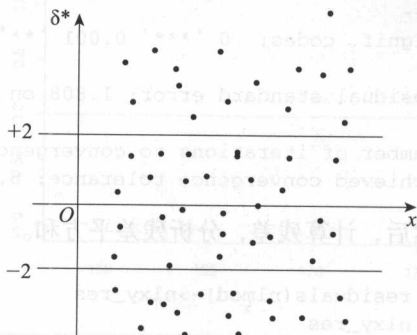


图 9-24 拟合不充分

根据标准残差图的相关知识, 观察图 9-19 所示的身高体重标准残差图, 可得出以下结论:

- 绝大部分数据在  $(-2, +2)$  的水平带状区间内, 因此模型拟合较充分。
- 数据点分布稍均匀, 但没有达到随机均匀分布的状态。此外, 部分数据点还是呈现某种曲线波动形状, 有少许系统性偏差, 因此可能采用非线性拟合效果会更好。

3) 大学生身高与体重非线性拟合。继续以女大学生身高与体重的例子为例, 再次仔细观察图 9-16 所示的散点图, 数据点呈现上凸的递增曲线分布, 可对身高与体重的关系应用非线性模型, 也许效果会比直线好, 因为曲线更贴近它的分布趋势。那么选用哪个非线性数学函数比较适合呢? 观察前面列举的常见函数图像, 可发现幂函数比较适合, 尤其是  $y=x^{1/2}$  的函数图像, 也呈现上凸且递增趋势, 但是幂函数  $y=x^a$  仅有一个参数  $a$ , 需要进行变形, 在函数尾部加上截距  $b$ , 使函数图像能在  $Y$  轴整体上下移动, 这样, 就拥有了两个可调节参数, 形成的拟合方程更灵活和适用。

细心的读者肯定发现了, 加上截距后, 幂函数  $y=x^a$  变成了多项式函数  $y=x^a+b$ , 因此将非线性拟合模型假设为  $y=x^a+b$  的多项式方程。

首先, 显示自变量和因变量数据。

> x

[1] 160 165 158 172 159 176 160 162 171

```
> y
[1] 58 63 57 65 62 66 58 59 62
```

接着,通过 `nlm` 函数建立非线性拟合模型,并通过 `summary` 函数分析该模型。

```
> nlm(y ~ Const + x^A)->nlmod
> summary(nlmod)
```

```
Formula: y ~ Const + x^A
```

```
Parameters:
```

```
      Estimate Std. Error t value Pr(>|t|)
Const -19.66594   15.09467  -1.303    0.234
A       0.86036    0.03657  23.523 6.37e-08 ***
```

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 1.808 on 7 degrees of freedom
```

```
Number of iterations to convergence: 4
```

```
Achieved convergence tolerance: 5.78e-06
```

然后,计算残差,分析残差平方和。

```
> residuals(nlmod)->nlxy_res
> nlxy_res
[1] -1.0986204  1.7882691 -1.2508068  0.8448399  3.3251002  0.1704208
-1.0986204 -0.9449555 -1.7357098
attr(,"label")
[1] "Residuals"
> sum(nlxy_res*nlxy_res)
[1] 22.88108
```

上述代码计算得出的残差平方和为 22.88108,比刚才应用线性拟合的残差平方和 22.88334 稍小些,可见非线性模型更适合拟合本例中的身高体重数据。

下面,绘制残差的 QQ 图,如图 9-25 所示。从 QQ 图中,可看出:残差  $\delta$  分布接近正态分布,数据点基本在直线附近。

```
> qqnorm(nlxy_res)
> qqline(nlxy_res)
```

接着,计算标准残差,将计算结果存入 `std_nlxy_res`。

```
> (nlxy_res-mean(nlxy_res))/sd(nlxy_res)->std_nlxy_res
> std_nlxy_res
[1] -0.6496072  1.0574063 -0.7395947  0.4995580  1.9661322  0.1007751
[7] -0.6496072 -0.5587453 -1.0263171
attr(,"label")
[1] "Residuals"
```

绘制标准残差图,如图 9-26 所示。

```
> plot(x,std_nlxxy_res)
> abline(h=0)
```

通过绘制图 9-26 所示的标准残差图,应用 summary 函数对非线性模型进行分析,以及计算残差平方和与标准残差,可得出以下结论:

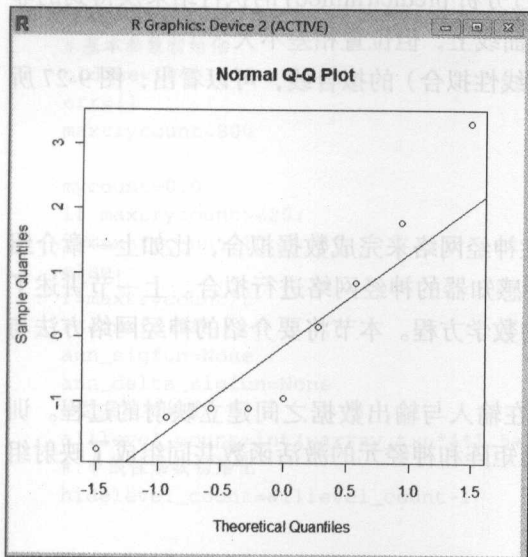


图 9-25 残差的 QQ 图

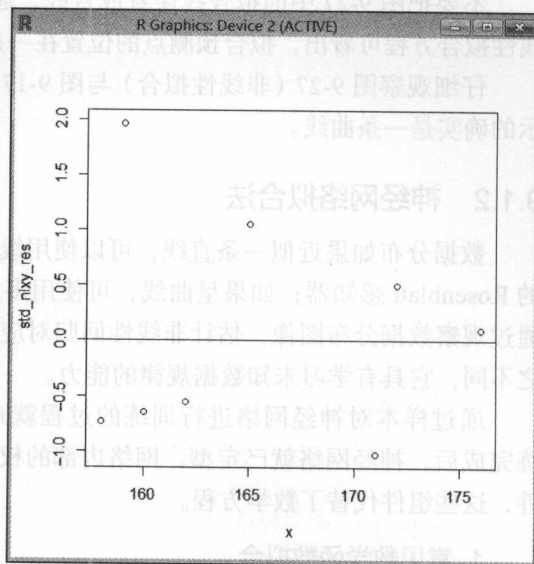


图 9-26 身高体重非线性模型标准残差图

- 图 9-26 中没有任何异常点,计算的标准残差全在  $(-2, +2)$  的水平带状区间内,模型拟合充分。
- 图 9-26 中数据点分布较均匀,拟合效果良好。
- 残差平方和比线性模型小,非线性模型更适于描述女大学生身高与体重的关系。
- 非线性拟合模型为  $y = x^{0.86036} - 19.66594$ 。

最后,观察一下拟合预测数据以及拟合效果图,如图 9-27 所示。

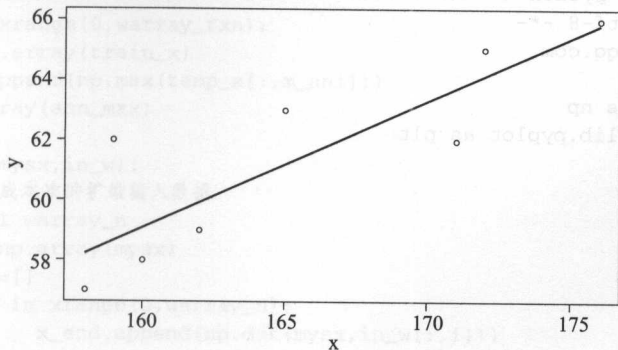


图 9-27 拟合效果图



```
> predict(nlmod)
[1] 59.09862 61.21173 58.25081 64.15516 58.67490 65.82958 59.09862 59.94496
[9] 63.73571
> plot(x,y)
> lines(x, predict(nlmod), col = 2)
```

不要把图 9-27 中的拟合线误看成直线，通过分析 `predict(nlmod)` 的执行结果及得到的非线性拟合方程可看出，拟合预测点的位置在一条曲线上，但位置相差不大。

仔细观察图 9-27（非线性拟合）与图 9-17（线性拟合）的拟合线，可以看出，图 9-27 所示的确实是一条曲线。

### 9.1.2 神经网络拟合法

数据分布如果近似一条直线，可以使用线性神经网络来完成数据拟合，比如上一章介绍的 Rosenblatt 感知器；如果呈曲线，可使用多层感知器的神经网络进行拟合。上一节讲述了通过观察数据分布图像，估计非线性回归对应的数学方程。本节将要介绍的神经网络方法与之不同，它具有学习未知数据规律的能力。

通过样本对神经网络进行训练的过程就是在输入与输出数据之间建立映射的过程。训练完成后，神经网络就已定型，网络内部的权值矩阵和神经元的激活函数共同组成了映射组件，这些组件代替了数学方程。

#### 1. 常见数学函数拟合

理论上来说，有了神经元作为映射组件，神经网络可以对数据之间的任何规律进行学习和模仿。下面是神经网络对上一节介绍的几个数学函数进行拟合的效果。

1) `sin` 函数拟合。顾名思义，`sin` 函数拟合就是使用一组数据  $x$  作为输入，通过神经网络建立模型，其输出值为  $\sin(x)$ 。可通过使用 Python 编写神经网络，实现 `sin` 函数拟合。

在华章网站下载本书的资源包，打开里面的文档“多层感知器神经网络源代码.doc”。下面将在该文档中源代码的基础上进行修改，实现非线性拟合。

首先，随机生成 500 个  $x$  值，同时计算这些样本对应的目标值。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-1.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy

isdebug=False
#x 和 d 样本初始化
train_x = []
d = []
for yb_i in xrange(0,500):
    train_x.append([np.random.rand()*4*np.pi-2*np.pi])
```

```
for yb_i in xrange(0,500):
    d.append(np.sin(train_x[yb_i]))
```

然后设置相关参数。代码如下:

```
warray_tnx=len(train_x[0])
warray_n=warray_tnx*4*2

# 基本参数初始化
oldmse=10**100
err=[]
maxtrycount=800

mycount=0.0
if maxtrycount>=20:
    r=maxtrycount/10
else:
    r=maxtrycount/2
#sigmoid 函数
ann_sigfun=None
ann_delta_sigfun=None
# 总层数初始化
alllevel_count=int(warray_tnx*4*1.5+1)
# 非线性层数初始化
hidelevel_count=alllevel_count-1

# 学习率参数
learn_r0=0.002
learn_r=learn_r0 *1.5
# 动量参数
train_a0=learn_r0*1.2
train_a0*=0.001
train_a=train_a0
expect_e=0.02
```

接着,对数据进行预处理,并生成初始权值矩阵。

# 对输入数据进行预处理

```
ann_max=[]
for m_ani in xrange(0,warray_tnx):
    temp_x=np.array(train_x)
    ann_max.append(np.max(temp_x[:,m_ani]))
ann_max=np.array(ann_max)

def getnowsx(mysx, in_w):
    ''' 生成本次的扩维输入数据 '''
    global warray_n
    mysx=np.array(mysx)
    x_end=[]
    for i in xrange(0,warray_n):
        x_end.append(np.dot(mysx, in_w[:,i]))
    return x_end
```

```

def get_inlw(my_train_max,w_count,myin_x):
    ''' 计算对输入数据预处理的权值 '''
    # 对随机生成的多个权值进行优化选择, 选择最优的权值
    global warray_txn
    global warray_n
    mylw=[]
    y_in=[]
    # 生成测试权值
    mylw=np.random.rand(w_count,warray_txn,warray_n)
    for ii in xrange(0,warray_txn):
        mylw[:,ii,:]=mylw[:,ii,:]*1/float(my_train_max[ii])-1/float(my_
            train_max[ii])*0.5

    # 计算输出
    for i in xrange(0,w_count):
        y_in.append([])
        for xj in xrange(0,len(myin_x)):
            y_in[i].append(getnowsx(myin_x[xj],mylw[i]))

    # 计算均方差
    mymin=10**5
    mychoice=0
    for i in xrange(0,w_count):
        myvar=np.var(y_in[i])
        if abs(myvar-1)<mymin:
            mymin=abs(myvar-1)
            mychoice=i

    # 返回数据整理的权值矩阵
    return mylw[mychoice]

mylnnw=get_inlw(ann_max,300,train_x)

def get_inputx(mytrain_x,myin_w):
    ''' 将训练数据通过输入权数, 扩维后形成输入数据 '''
    end_trainx=[]
    for i in xrange(0,len(mytrain_x)):
        end_trainx.append(getnowsx(mytrain_x[i],myin_w))
    return end_trainx

x=get_inputx(train_x,mylnnw)

def get_siminx(sim_x):
    global mylnnw
    myxx=np.array(sim_x)
    return get_inputx(myxx,mylnnw)

def getlevelw(myin_x,wo_n,wi_n,w_count):
    ''' 计算一层的初始化权值矩阵 '''
    mylw=[]
    y_in=[]
    # 生成测试权值
    mylw=np.random.rand(w_count,wi_n,wo_n)
    mylw=mylw*2.-1

    # 计算输出

```

```

for i in xrange(0,w_count):
    y_in.append([])
    for xj in xrange(0,len(myin_x)):
        x_end=[]
        for myii in xrange(0,wo_n):
            x_end.append(np.dot(myin_x[xj],mylw[i,:,myii]))
        y_in[i].append(x_end)
# 计算均方差
mymin=10**3
mychoice=0
for i in xrange(0,w_count):
    myvar=np.var(y_in[i])
    if abs(myvar-1)<mymin:
        mymin=abs(myvar-1)
        mychoice=i
# 返回数据整理的权值矩阵
csmylw=mylw[mychoice]
return csmylw,y_in[mychoice]

ann_w=[]
def init_annw():
    global x
    global hidelevel_count
    global warray_n
    global d
    global ann_w
    ann_w=[]

    lwyii=np.array(x)
    for myn in xrange(0,hidelevel_count):
        # 层数
        ann_w.append([])
        if myn==hidelevel_count-1:
            for iii in xrange(0,warray_n):
                ann_w[myn].append([])
                for jjj in xrange(0,warray_n):
                    ann_w[myn][iii].append(0.0)
        elif myn==hidelevel_count-2:
            templtw,lwyii=getlevelw(lwyii,len(d[0]),warray_n,10)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])
                for xjj in xrange(0,len(d[0])):
                    ann_w[myn][xii].append(templtw[xii,xjj])
                for xjj in xrange(len(d[0]),warray_n):
                    ann_w[myn][xii].append(0.0)
        else:
            templtw,lwyii=getlevelw(lwyii,warray_n,warray_n,10)
            for xii in xrange(0,warray_n):
                ann_w[myn].append([])
                for xjj in xrange(0,warray_n):
                    ann_w[myn][xii].append(templtw[xii,xjj])
    ann_w=np.array(ann_w)

def generate_lw(trycount):

```

```

global ann_w
print u" 产生权值初始矩阵 ",
meanmin=1
myann_w=ann_w
alltry=30
tryc=0
while tryc<alltry:
    for i_i in range(trycount):
        print ".",
        init_annw()
        if abs(np.mean(np.array(ann_w)))<meanmin:
            meanmin=abs(np.mean(np.array(ann_w)))
            myann_w=ann_w
        tryc+=1
        if abs(np.mean(np.array(myann_w)))<0.01:break
ann_w=myann_w
print
print u" 权值矩阵平均 :%f"%(np.mean(np.array(ann_w)))
print u" 权值矩阵方差 :%f"%(np.var(np.array(ann_w)))

generate_lw(15)

```

下面，对所有样本数据进行反复训练，直到误差降到期望值为止。单个样本的训练过程如下：

```

def sample_train(myx,myd,n,sigmoid_func,delta_sigfun):
    ''' 一个样本的前向和后向计算 '''

```

```

global ann_yi
global ann_delta
global ann_w
global ann_wj0
global ann_y0
global hidelevel_count
global alllevel_count
global learn_r
global train_a
global ann_oldw

level=hidelevel_count
# 清空 yi 输出信号数组
hidelevel=hidelevel_count
alllevel=alllevel_count
for i in xrange(0,alllevel):
    # 第一维是层数，从 0 开始
    for j in xrange(0,n):
        # 第二维是神经元
        ann_yi[i][j]=0.0
ann_yi=np.array(ann_yi)
yi=ann_yi

```



```

# 清空 delta 矩阵
for i in xrange(0,hidelevel-1):
    for j in xrange(0,n):
        ann_delta[i][j]=0.0

delta=ann_delta

# 保留 w 的拷贝，以便下一次迭代
ann_oldw=copy.deepcopy(ann_w)
oldw=ann_oldw

# 前向计算
if isdebug:print u" 前向计算中 ..."
# 对输入变量进行预处理
myo=np.array([])
for nowlevel in xrange(0,alllevel):
    # 一层层向前计算
    # 计算诱导局局部域
    my_y=[]
    myy=yi[nowlevel-1]
    myw=ann_w[nowlevel-1]
    if nowlevel==0:
        # 第一层隐藏层
        my_y=myx
        yi[nowlevel]=my_y
    elif nowlevel==(alllevel-1):
        # 输出层
        my_y=o_func(yi[nowlevel-1,:len(myd)])
        yi[nowlevel,:len(myd)]=my_y
    elif nowlevel==(hidelevel-1):
        # 最后一层输出层
        for i in xrange(0,len(myd)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel,:len(myd)]=my_y
    else:
        # 中间隐藏层
        for i in xrange(0,len(myy)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel]=my_y

if isdebug:
    print u"***** 本样本训练的输出矩阵 *****"
    print yi
    print u"*****"

# 计算误差与均方误差
# 因为线性输出层为直接复制，所以取非线性隐藏输出层的结果
myo=yi[hidelevel-1][:len(myd)]
myo_end=yi[alllevel-1][:len(myd)]
mymse=get_e(myd,myo_end)

```

```

# 反向计算
# 输入层不需要计算 delta, 输出层不需要计算 w
if isdebug: print u" 反向计算中 ..."

# 计算 delta
for nowlevel in xrange(level-1, 0, -1):
    if nowlevel==level-1:
        mydelta=delta[nowlevel]
        my_n=len(myd)
    else:
        mydelta=delta[nowlevel+1]
        my_n=n
    myw=ann_w[nowlevel]
    if nowlevel==level-1:
        # 输出层
        mydelta=delta_sigfun(myo, myd, None, None, None, None, None)
        mydelta=mymse*myo
    elif nowlevel==level-2:
        # 输出隐藏层的前一层, 因为输出结果和前一隐藏层的神经元数目可能存在
        # 不一致的情况, 所以单独处理, 传输相当于输出隐藏层的神经元数目的数据
        mydelta=delta_sigfun(yi[nowlevel], myd, nowlevel, level-
            1, my_n, mydelta[:len(myd)], myw[:, :len(myd)])
    else:
        mydelta=delta_sigfun(yi[nowlevel], myd, nowlevel, level-
            1, my_n, mydelta, myw)

    delta[nowlevel][:my_n]=mydelta

# 计算与更新权值 w
for nowlevel in xrange(level-1, 0, -1):
    # 每个层的权值不一样
    if nowlevel==level-1:
        # 输出层
        my_n=len(myd)
        mylearn_r=learn_r*0.8
        mytrain_a=train_a*1.8
    elif nowlevel==1:
        # 输入层
        my_n=len(myd)
        mylearn_r=learn_r*0.9
        mytrain_a=train_a*0.8
    else:
        # 其他层
        my_n=n
        mylearn_r=learn_r
        mytrain_a=train_a

    pre_level_myy=yi[nowlevel-1]
    pretrain_myww=oldw[nowlevel-1]
    pretrain_myw=pretrain_myww[:, :my_n]

# 第二个调整参数
temp_i=[]

```

```

for i in xrange(0,n):
    temp_i.append([])
    for jj in xrange(0,my_n):
        temp_i[i].append(mylearn_r*delta[nowlevel,jj]*pre_
            level_myy[i])
    temp_rs2=np.array(temp_i)
    temp_rs1=mytrain_a*pretrain_myy
    # 总调整参数
    temp_change=temp_rs1+temp_rs2
    my_wv=ann_w[nowlevel-1]
    my_wv[:, :my_n]+=temp_change
    if isdebug:
        print "=====
        print u"*** 权值矩阵 ***
        print ann_w
        print u"*** 梯度矩阵 ***
        print delta
        print "=====
    return mymse

```

经过 43 次训练，误差率降到了 0.02 以下，训练过程停止，如下所示：

```

.....
.....
----- 开始第 41 次训练 ----- 误差为：0.026790
----- 开始第 42 次训练 ----- 误差为：0.021292
----- 开始第 43 次训练 ----- 误差为：0.019350
训练成功，正在进行检验

```

最后，进行仿真测试。下面是一个样本值的仿真计算过程。

```

def simulate(myx, sigmoid_func, delta_sigfun):

```

```

    ''' 一个样本的仿真计算 '''
    print u" 仿真计算中 "
    global ann_yi
    global ann_w
    global ann_wj0
    global ann_y0
    global hidelevel_count
    global alllevel_count
    global d
    global mylnwv

```

```

    myd=d[0]
    myx=np.array(myx)
    n=len(myx)

    # 清空 yi 输出信号数组
    hidelevel=hidelevel_count
    alllevel=alllevel_count
    for i in xrange(0,alllevel):
        # 第一维是层数，从 0 开始
        for j in xrange(0,n):
            # 第二维是神经元

```

```

ann_yi[i][j]=0.0
ann_yi=np.array(ann_yi)
yi=ann_yi

# 前向计算
myy=np.array([])
for nowlevel in xrange(0,alllevel):
    # 一层层向前计算
    # 计算诱导局部域
    my_y=[]
    myy=yi[nowlevel-1]
    myw=ann_w[nowlevel-1]
    if nowlevel==0:
        # 第一层隐藏层
        my_y=myx
        yi[nowlevel]=my_y
    elif nowlevel==(alllevel-1):
        # 线性输出层
        my_y=o_func(yi[nowlevel-1,:len(myd)])
        yi[nowlevel,:len(myd)]=my_y
    elif nowlevel==(hidelevel-1):
        # 最后一层隐藏输出层
        for i in xrange(0,len(myd)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel,:len(myd)]=my_y
    else:
        # 中间隐藏层
        # 中间隐藏层需要加上偏置
        for i in xrange(0,len(myy)):
            temp_y=sigmoid_func(np.dot(myw[:,i],myy))
            my_y.append(temp_y)
        yi[nowlevel]=my_y

if isdebug:
    print "======"
    print u"*** 权值矩阵 ***"
    print ann_w
    print u"*** 输出矩阵 ***"
    print yi
    print "======"
return yi[alllevel-1,:len(myd)]

```

为了验证效果，绘制数据拟合效果和误差曲线，如图 9-28 所示。图 9-28 的上部是拟合效果图，目标值和预测值很接近，下部是误差曲线，下降平滑。

2)  $0.6\sin(x)$  函数神经网络拟合。下面尝试实现稍复杂数学函数的拟合，比如  $y=0.6\sin(x)$ 。这里的 Python 实现代码与前面相同，因此，只是在样本数据初始化代码段进行了修改。示例如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-

```

```
#code:myhaspl@qq.com
#9-2.py
import numpy as np
import matplotlib.pyplot as plt
import random
import copy

isdebug=False
#x和d样本初始化
train_x=[]
d=[]
for yb_i in xrange(0,500):
    train_x.append([np.random.rand()*4*np.pi-2*np.pi])
for yb_i in xrange(0,500):
    d.append(np.sin(train_x[yb_i])*0.6)
```

经过 71 次训练, 神经网络的收敛目标达到, 也就是说误差率达到了期望值。

```
.....
----- 开始第 69 次训练 ----- 误差为: 0.020464
----- 开始第 70 次训练 ----- 误差为: 0.020673
----- 开始第 71 次训练 ----- 误差为: 0.019350
训练成功, 正在进行检验
```

图 9-29 为程序绘制的拟合效果和误差曲线图。

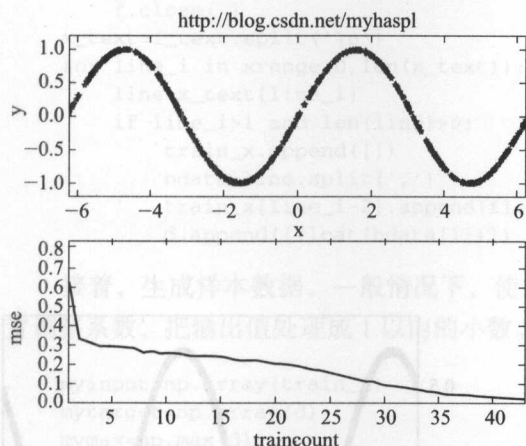


图 9-28  $\sin$  函数神经网络拟合

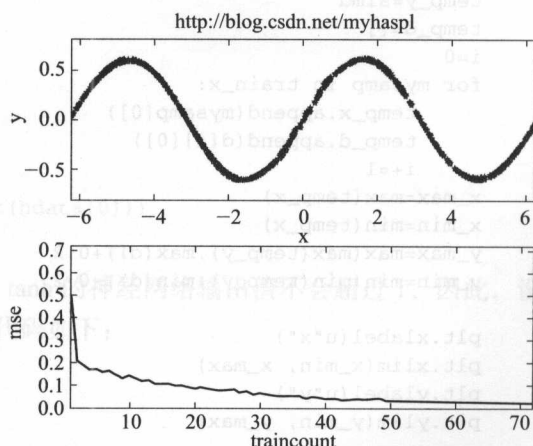


图 9-29  $0.6\sin(x)$  函数神经网络拟合 (附彩图)

3)  $0.5\sin(x)+0.5\cos(x)$  函数拟合。前两个例子使用了本书资源包所示的 Python 代码。下面使用 Neurolab 库实现对  $0.5\sin(x)+0.5\cos(x)$  的拟合。代码如下:

```
#!/usr/bin/env python
-*- coding: utf-8 -*-
#9-3.py
#拟合  $\sin*0.5+\cos*0.5$ 
```



```

import neurolab as nl
import numpy as np
import matplotlib.pyplot as plt
isdebug=False

#x和d样本初始化
train_x=[]
d=[]
samplescount=1000
myrndsmp=np.random.rand(samplescount)
for yb_i in xrange(0,samplescount):
    train_x.append([myrndsmp[yb_i]*4*np.pi-2*np.pi])
for yb_i in xrange(0,samplescount):
    d.append(np.sin(train_x[yb_i])*0.5+np.cos(train_x[yb_i])*0.5)

myinput=np.array(train_x)
mytarget=np.array(d)

bpnet = nl.net.newff([[-2*np.pi, 2*np.pi]], [5, 1])
err = bpnet.train(myinput, mytarget, epochs=800, show=100, goal=0.02)

simd=[]
for xn in xrange(0,len(train_x)):
    simd.append(bpnet.sim([train_x[xn]])[0][0])

temp_x=[]
temp_y=simd
temp_d=[]
i=0
for mysamp in train_x:
    temp_x.append(mysamp[0])
    temp_d.append(d[i][0])
    i+=1
x_max=max(temp_x)
x_min=min(temp_x)
y_max=max(max(temp_y),max(d))+0.2
y_min=min(min(temp_y),min(d))-0.2

plt.xlabel(u"x")
plt.xlim(x_min, x_max)
plt.ylabel(u"y")
plt.ylim(y_min, y_max)

lp_x1 = temp_x
lp_x2 = temp_y
lp_d = temp_d
plt.plot(lp_x1, lp_x2, 'r*')
plt.plot(lp_x1,lp_d,'bo')
plt.show()

```

800次训练后，拟合效果较好，如图9-30所示。

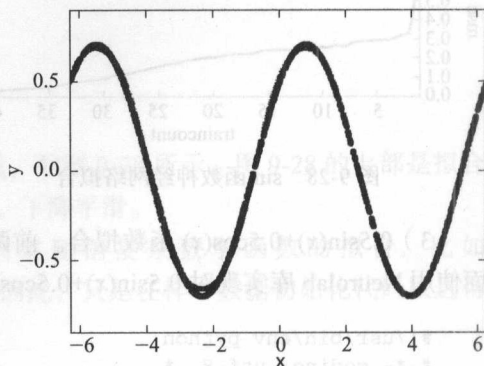


图 9-30  $0.5\sin(x)+0.5\cos(x)$  的拟合

## 2. 钢包使用次数与容积模型拟合

表 9-3 是钢包使用次数与容积实测数据, 以  $x$  为输入,  $y$  为输出, 在输入与输出数据之间可建立非线性关系。这里用神经网络建立数据拟合模型。

下面尝试应用神经网络完成以上数据拟合任务, 这里调用 Neurolab 库, 用 Python 实现。

首先读取数据文件。代码如下:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-4.py
import numpy as np
import pylab as pl
import neurolab as nl
```

```
print u'正在处理中'
```

```
#x 和 d 样本初始化
```

```
train_x = []
```

```
d = []
```

```
f = open("cubage.csv")
```

```
try:
```

```
    f_text = f.read()
```

```
finally:
```

```
    f.close()
```

```
x_text=f_text.split('\n')
```

```
for line_i in xrange(0,len(x_text)):
```

```
    line=x_text[line_i]
```

```
    if line_i>1 and len(line)>0:
```

```
        train_x.append([])
```

```
        hdata=line.split(',')
```

```
        train_x[line_i-2].append(float(hdata[0]))
```

```
        d.append([float(hdata[1])])
```

表 9-3 钢包使用次数与容积实测数据

使用次数 $x$	容积 $y$
2	106.42
3	108.2
4	109.58
5	109.5
7	110
8	109.93
10	110.49
11	110.59
14	110.6
15	110.9
16	110.7
18	111
19	111.2

接着, 生成样本数据, 一般情况下, 使用  $\tanh$  的神经网络输出值不会超过 1, 因此, 设置调整系数, 把输出值处理成 1 以内的小数。代码如下:

```
myinput=np.array(train_x)
```

```
mytarget=np.array(d)
```

```
mymax=np.max(d)
```

```
tz=(0.1**(len(str(int(mymax)))))*5
```

```
myinput=myinput
```

```
mytarget=tz*mytarget
```

最后进行训练, 训练时可加入未知样本进行测试。

```
# 对未知样本进行测试
```

```
myinputtest=[[6],[9],[17],[20]]
```

```
testsimd= bpnet.sim(myinputtest)
```

```
end_x=np.array(myinputtest)
```

```
testsimd/=tz
end_y=testsimd
```

经过 9 次训练，达到了训练目标，误差为  $1.9713682268777952e-05$ 。对神经网络输出值应用调整系数，将其输出值恢复到原有的数值范围。

程序生成了拟合效果图（如图 9-31 上部所示）及误差曲线图（如图 9-31 下部所示）。同时使用在样本空间中没有出现的部分测试数据 6、9、17、20 作为神经网络的输入，验证神经网络的训练效果和拟合效果。图中实心圆圈为未知测试数据，即：使用次数为 6、9、17、20 时容积的预测值，星号为样本数据。

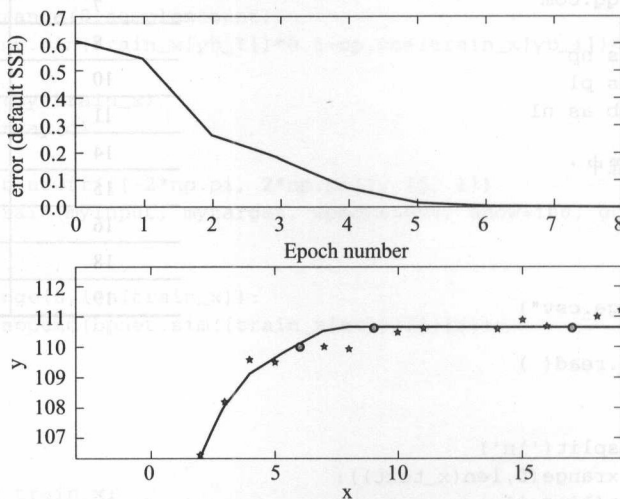


图 9-31 钢包使用次数与容积的神经网络拟合

从图 9-31 的拟合效果来看，样本数据非常接近神经网络拟合的曲线，说明曲线较好地反映了随着使用次数的增加，容积的增长趋势。此外，对在样本中没出现的钢包使用次数 6、9、17、20，与之相关的容积预测效果不错。

完整的 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-4.py
import numpy as np
import pylab as pl
import neurolab as nl
print u'正在处理中'
#x 和 d 样本初始化
train_x = []
d = []
f = open("cubage.csv")
try:
    f_text = f.read()
```

```

finally:
    f.close()
x_text=f_text.split('\n')
for line_i in xrange(0,len(x_text)):
    line=x_text[line_i]
    if line_i>1 and len(line)>0:
        train_x.append([])
        hdata=line.split(',')
        train_x[line_i-2].append(float(hdata[0]))
        d.append([float(hdata[1])])
myinput=np.array(train_x)
mytarget=np.array(d)
mymax=np.max(d)
tz=(0.1**(len(str(int(mymax)))))*5
myinput=myinput
mytarget=tz*mytarget
netminmax=[0,np.max(myinput)]
print u'\n 正在建立神经网络'
bpnet = nl.net.newff([netminmax], [5, 1])

print u'\n 训练神经网络中 ...'
err = bpnet.train(myinput, mytarget, epochs=800, show=5, goal=0.0001)
if err[len(err)-1]>0.0001:
    print u'\n 训练神经网络失败 ...'\n'
else:
    print u'\n 训练神经网络完毕'
    pl.subplot(211)
    pl.plot(err)
    pl.xlabel('Epoch number')
    pl.ylabel('error (default SSE)')
    # 对样本进行测试
    simd= bpnet.sim(myinput)
    temp_x=myinput
    temp_d=mytarget
    simd/=tz
    temp_y=simd
    temp_d/=tz

    # 对未知样本进行测试
    myinputtest=[[6],[9],[17],[20]]
    testsimd= bpnet.sim(myinputtest)
    end_x=np.array(myinputtest)
    testsimd/=tz
    end_y=testsimd

    x_max=np.max(temp_x)
    x_min=np.min(temp_x)-5
    y_max=np.max(temp_y)+2
    y_min=np.min(temp_y)

    pl.subplot(212)
    pl.xlabel(u"x")
    pl.xlim(x_min, x_max)

```

```

pl.ylabel(u"y")
pl.ylim(y_min, y_max)
lp_x1 = temp_x
lp_x2=temp_y
lp_d = temp_d
pl.plot(lp_x1, lp_x2, 'g-')
pl.plot(end_x, end_y, 'ro')
pl.plot(lp_x1,lp_d,'b*')

```

## 9.2 线性滤波

### 9.2.1 WAV 声音文件

声音是由物体的机械振动而形成的。用鼓棒敲击鼓皮，于是鼓皮发生振动而发声；吹笛时笛腔内的空气柱发生振动而发声；把音频电流送入扬声器，扬声器的纸盆发生振动而发声。发生声音的振动源叫作“声源”，由声源发出的声音，必须通过媒质才能传送到我们的耳朵中。空气是最常见的媒质，如水、金属、木材等媒质都能传播声音。

WAV 声音文件为 Microsoft 公司开发的一种记录声音的文件格式，它符合 RIFF (Resource Interchange File Format) 文件规范，音频格式未经过压缩，在音质方面不会出现失真的情况。它以指定频率采样，比如每秒采样 44100 次。在采集声音的振动状态时，它会使用模 / 数转换器 (A/D) 以每秒上万次的速率对声波进行采样，每一次采样都记录下了原始模拟声波在某一时刻的状态，采样的结果即为样本。将一串样本连接起来，就可以描述一段声波了。

### 9.2.2 线性滤波算法过程

神经网络既可以进行函数拟合，也能对波形数据进行拟合。下面将输入  $x$  作为函数的自变量，将神经网络输出的数据  $y$  作为函数  $f(x)$  的输出，然后应用该拟合功能进行一个线性滤波，以实现去除周围环境的噪音，使说话声音更清楚。

下面以从含有音乐的语音中去除背景音乐为例进行讲解。具体算法过程如下：

1) 读取该机器学习算法必需的两个素材文件：含有背景音乐的语音和部分背景音乐。细心的读者一定会问，既然已经有背景音乐这个文件，那么直接从语音的波形数据减去背景音乐，这样就完成了背景音乐的过滤。没错，但是不能直接减去源背景音乐。原因如下：

在对声波进行采样时，虽然同时对背景音乐和语音进行了采样，但采样过程很可能会被周边的音源、采样音量等很多因素干扰，导致采样形成的背景音乐波形并不是源背景音乐的波形。如果直接使用源背景音乐进行减，将会导致滤波后的语音文件有杂音且部分失真。我们要做的是干净地将背景音乐剔除。

2) 采样器先采样一小段背景音乐，然后再采集语音文件。其好处在于：生成了经过采样后的背景音乐样本，这些样本就是神经网络拟合训练样本中的目标输出值。



3) 用神经网络进行训练, 输入样本为源背景音乐中相当于采样背景音乐长度的波形数据, 输出目标为采样器开始采集的小段背景音乐波形数据。

4) 训练达到期望误差率或达到最大训练次数后, 将源背景音乐作为未知样本数据送入神经网络, 其预测输出就是拟合后的采样背景音乐。

5) 将混杂有背景音乐的语音文件波形数据直接减去拟合后的采样背景音乐, 得到去除背景音乐的纯净语音。

自适应线性滤波器的工作原理也是如此: 先采集一段背景噪声; 然后用噪声数据进行拟合, 之后在噪声数据和背景噪声之间建立了一种映射关系; 最后, 人可以通过该系统传送语音, 说话周围环境产生的背景噪声将被过滤。

### 9.2.3 滤波 Python 实现

下面用 Python 实现上述步骤。

1) 读取声音素材, 代码如下:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-5.py
import numpy as np
import wave
import pylab as pl
import copy
print 'working...'

print "read wav data...."
err=[]
# 打开 WAV 文档
f = wave.open(r"speak.wav", "rb")
fo = wave.open(r"wait_jg.wav", "wb")
fi=wave.open(r"back.wav", "rb")
fend=wave.open(r"end_jg.wav", "wb")

# 读取波形数据
# (nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)

fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)

# 将波形数据转换为数组, 并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)
```

2) 复制波形, 在一个随机的基数基础上, 将背景音乐的振幅(音量)稍微变动一下, 并与语音合并。前面预留一段经过波形调整后的背景音乐, 以供线性神经网络拟合用。

```
emptywdata=np.zeros(framerate, dtype=np.short)
new_wave_data=np.hstack((emptywdata,wave_data,wave_data,wave_data,wave_
data,wave_data,wave_data,wave_data,wave_data))
wave_data =copy.deepcopy(new_wave_data)
nframes*=8
nframes+=framerate/2
temp_wavedata=np.hstack((fi_wave_data,fi_wave_data))[:len(new_wave_data)]
backrnd=np.random.rand(len(new_wave_data))*10-5
backbase=np.random.rand()*2+1
temp_wavedata=temp_wavedata*backbase+backrnd
new_wave_data=temp_wavedata+new_wave_data

new_wave_data=np.array(new_wave_data)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
```

3) 拟合数据, 去除背景音乐。

```
jg_wave_data=copy.deepcopy(new_wave_data)
jg_temp_wavedata=np.hstack((fi_wave_data,fi_wave_data))[:len(new_wave_data)]
jg_temp_wavedata=jg_temp_wavedata[:len(new_wave_data)]*w[1]+w[0]
jg_wave_data=jg_wave_data-jg_temp_wavedata

for jg_i in xrange(0,len(jg_wave_data)):
    if abs(jg_wave_data[jg_i])<500:
        jg_wave_data[jg_i]=0
jg_wave_data[:framerate]=0

jg_wave_data =jg_wave_data.astype(wave_data.dtype)
jg_str_data=jg_wave_data.tostring()

print "save output wav...."
fend.setnchannels(nchannels)
fend.setframerate(framerate)
fend.setsampwidth(sampwidth)
fend.writeframes(jg_str_data)
```

完整的代码如下:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#9-5.py
import numpy as np
import wave
import pylab as pl
import copy
print 'working...'

print "read wav data...."
```

```

err=[]
# 打开 WAV 文档
f = wave.open(r"speak.wav", "rb")
fo = wave.open(r"wait_jg.wav", "wb")
fi=wave.open(r"back.wav", "rb")
fend=wave.open(r"end_jg.wav", "wb")

# 读取波形数据
# (nchannels, sampwidth, framerate, nframes, comptype, compname)
params = f.getparams()
nchannels, sampwidth, framerate, nframes = params[:4]
str_data = f.readframes(nframes)

fi_params=fi.getparams()
fi_nframes = fi_params[3]
fi_str_data=fi.readframes(fi_nframes)

# 将波形数据转换为数组, 并更改
print "update wav data...."
wave_data = np.fromstring(str_data, dtype=np.short)
fi_wave_data= np.fromstring(fi_str_data, dtype=np.short)

# 复制并将背景音乐的振幅(音量)在一个随机的基数基础上稍微变动后, 与语音合并
# 前面预留一段经过波形调整后的背景音乐, 以供线性神经网络拟合用
emptywdata=np.zeros(framerate, dtype=np.short)
new_wave_data=np.hstack((emptywdata,wave_data,wave_data,wave_data,wave_
data,wave_data,wave_data,wave_data))
wave_data =copy.deepcopy(new_wave_data)
nframes*=8
nframes+=framerate/2
temp_wavedata=np.hstack((fi_wave_data,fi_wave_data))[:len(new_wave_data)]
backrnd=np.random.rand(len(new_wave_data))*10-5
backbase=np.random.rand()*2+1
temp_wavedata=temp_wavedata*backbase+backrnd
new_wave_data=temp_wavedata+new_wave_data

new_wave_data=np.array(new_wave_data)
new_wave_data =new_wave_data.astype(wave_data.dtype)
new_str_data=new_wave_data.tostring()
# 写波形数据参数
print "save mix wav files...."
fo.setnchannels(nchannels)
fo.setframerate(framerate)
fo.setsampwidth(sampwidth)
fo.writeframes(new_str_data)

# 线性逼近前段噪声
b=1
a0=5e-1
a=0.0
r=1.5
x=[]
d=[]

```

```

ii=0
for audio_i in xrange(0, framerate/2):
    if fi_wave_data[audio_i]!=0.:
        x.append([])
        x[ii].append(1)
        x[ii].append(fi_wave_data[audio_i])
        d.append(new_wave_data[audio_i])
        ii+=1
    if ii>100:
        break
x=np.array(x)
d=np.array(d)

w=np.random.rand(2)*np.mean(x)#np.array([b,0])
expect_e=15
maxtrycount=10000
mycount=0
def sgn(v):
    return v
def get_v(myw,myx):
    return sgn(np.dot(myw.T,myx))
def neww(oldw,myd,myx,a):
    mye=get_e(oldw,myx,myd)
    a=a0/(1+float(mycount)/r)
    return (oldw+a*mye*myx,mye)
def get_e(myw,myx,myd):
    return myd-get_v(myw,myx)

while True:
    mye=0.
    i=0
    for xn in x:
        w,e=neww(w,d[i],xn,a)
        i+=1
        mye+=pow(e,2)
    mye=np.sqrt(mye)
    mycount+=1
    err.append(mye)
    print u" 第 %d 次调整后的权值: "%mycount
    print w
    print u" 误差: %f"%mye
    if abs(mye)<expect_e or mycount>maxtrycount:break

print "w:[%f,%f]"%(w[0],w[1])

# 复制并除去背景声音
jg_wave_data=copy.deepcopy(new_wave_data)
jg_temp_wavedata=np.hstack((fi_wave_data,fi_wave_data))[:len(new_wave_data)]
jg_temp_wavedata=jg_temp_wavedata[:len(new_wave_data)]*w[1]+w[0]
jg_wave_data=jg_wave_data-jg_temp_wavedata

for jg_i in xrange(0,len(jg_wave_data)):
    if abs(jg_wave_data[jg_i])<500:

```

```
jg_wave_data[jg_i]=0
jg_wave_data[:framerate]=0
```

```
jg_wave_data =jg_wave_data.astype(wave_data.dtype)
jg_str_data=jg_wave_data.tostring()
```

```
print "save output wav...."
fend.setnchannels(nchannels)
fend.setframerate(framerate)
fend.setsampwidth(sampwidth)
fend.writeframes(jg_str_data)
```

# 绘制波形

```
time = np.arange(0, nframes) * (1.0 / framerate)
wave_data.shape = -1, 2
wave_data = wave_data.T
```

```
pl.subplot(321)
pl.plot(time, wave_data[0])
pl.subplot(322)
pl.plot(time, wave_data[1], c="g")
pl.xlabel("time (seconds)")
```

# 绘制波形

```
new_wave_data.shape = -1, 2
new_wave_data =new_wave_data.T
```

```
pl.subplot(323)
pl.plot(time,new_wave_data[0])
pl.subplot(324)
pl.plot(time, new_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()
```

# 绘制波形

```
jg_wave_data.shape = -1, 2
jg_wave_data =jg_wave_data.T
```

```
pl.subplot(325)
pl.plot(time,jg_wave_data[0])
pl.subplot(326)
pl.plot(time, jg_wave_data[1], c="g")
pl.xlabel("time (seconds)")
pl.show()
```

如图 9-32 所示是程序运行完毕后生成的效果图。最上面的两张图是无背景音乐语音文件的两个声道的波形图，中间是混杂了背景音乐的语音文件，下面是经过线性滤波后生成的纯净语音文件。

观察图 9-32 可看出，线性滤波的效果是不错的。读者可以从本书的资源包中下载相关程序，执行后，用音箱播放滤波后的语音，听听效果，是否还能听出背景音乐。



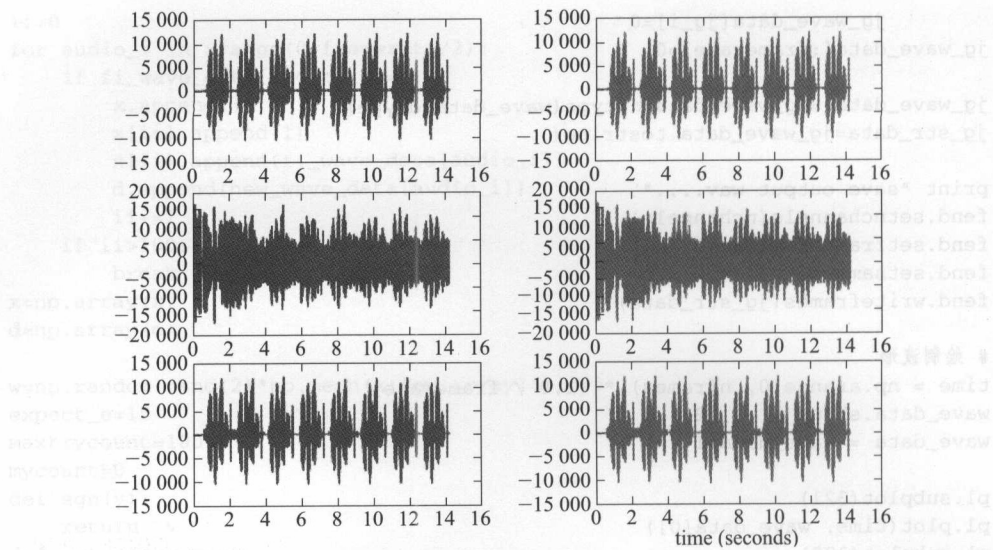


图 9-32 线性滤波效果

## 9.3 数据或曲线平滑

### 9.3.1 平滑概述

在介绍平滑的概念之前，先来看一组数据，表 9-4 是某虚拟商品 X 连续 21 天的销量。

表 9-4 虚拟商品 X 连续 21 天的销量

天	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
销量	12	123	54	176	121	134	198	155	122	111	133	244	278	232	267	222	187	193	245	110	132

现绘制该虚拟商品 X 的销量图（如图 9-33 所示），R 代码如下：

```
>
x<-c(12,123,54,176,121,134,198,155,122,111,133,244,278,232,267,222,187,193,245,110,132)
> plot(x,type="o",col="blue",xlab=" 天 ",ylab=" 销量 ")
```

初步观察图 9-33，曲线波折较多，销量不稳定，仔细观察才能发现销量在整体走高。再观察图 9-34，它是将图 9-33 平滑处理后生成的销量走势图。

观察图 9-34，曲线波动较小，整体增长趋势一目了然，销量连续经过了两次攀升，虽然每次攀升后有少量回落，但销量仍在增长中。

现在正式介绍平滑的概念，数据或曲线平滑是通过建立近似函数来发现数据中的主要模式的，去除噪音、结构细节或瞬时现象，减少不必要的波动，从而实现数据集的平滑。平滑过程中会修改数据点，降低由噪音数据点产生的影响，而低于毗邻数据点的点则被提升，从而得到一个更平滑的数据序列。

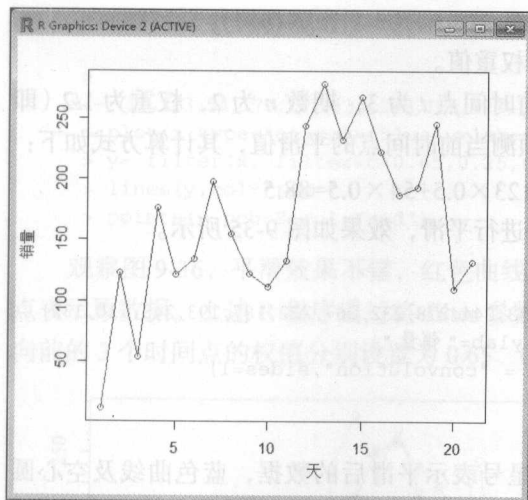


图 9-33 虚拟商品的销量图

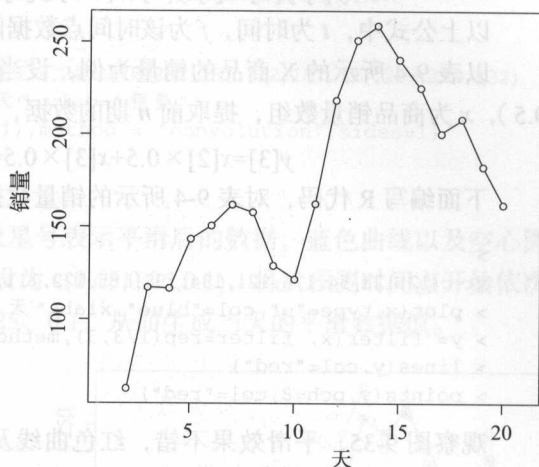


图 9-34 经过平滑处理后的销量走势图

### 9.3.2 移动平均

移动平均 (Moving Average, MA), 又称“移动平均线”, 简称均线, 是数据分析中一种分析时间序列数据的工具。最常见的是利用股价、交易量等变量计算出移动平均。移动平均可抚平短期波动, 反映出长期趋势或周期, 在数学上, 移动平均可视为一种卷积。

#### 1. 简单移动平均

简单移动平均是某变量之前  $n$  个数值的未作加权的算术平均。比如, 设某股票的预测收市价为  $SMA$ , 该股票前  $n$  日的收市价为  $p_1$  至  $p_n$ , 则预测收市价的计算方式如下:

$$SMA = \frac{p_1 + p_2 + \cdots + p_n}{n}$$

上述公式的计算效率较高, 原因在于: 计算连续的数值时, 若有一个新的数值加入, 可将一个旧的数值剔除, 无须每次都重新将数值逐个加起来, 如下面的公式所示:

$$SMA_{t1,n+1} = SMA_{t0,n} - \frac{p_1}{n} + \frac{p_{n+1}}{n}$$

在 R 语言中, 可调用 `filter` 函数进行线性过滤, 实现简单移动平均算法, 使用时需要注意以下事项:

- 1) `filter` 参数 (该参数指定了参加平均计算的前期数据的权重) 设定为  $1/n$ ,  $n$  为前  $n$  期。
- 2) `method` 参数指定为 `convolution`, 表示使用移动平均法, 并将 `sides` 参数设置为 1, 表示使用单边卷积。
- 3) `filter` 函数对数据进行预测时, 参与计算的前期数据不但包括前  $n-1$  期的数据, 还包括当前数据, 预测值的计算公式如下:

$$y[t]=f[1]*x[t]+f[2]*x[t-1]+f[3]*x[t-2]+\cdots+f[n-1]*x[t-(n-1)]$$

以上公式中,  $t$  为时间,  $f$  为该时间点数据的权重值。

以表 9-4 所示的 X 商品的销量为例, 设当前时间点  $t$  为 3, 期数  $n$  为 2, 权重为 1/2 (即 0.5),  $x$  为商品销量数组, 提取前  $n$  期的数据, 预测当前时间点的平滑值, 其计算方式如下:

$$y[3]=x[2]\times 0.5+x[3]\times 0.5=123\times 0.5+54\times 0.5=88.5$$

下面编写 R 代码, 对表 9-4 所示的销量数据进行平滑, 效果如图 9-35 所示。

```
>
x<-c(12,123,54,176,121,134,198,155,122,111,133,244,278,232,267,222,187,193,245,110,132)
> plot(x,type="o",col="blue",xlab=" 天",ylab=" 销量 ")
> y= filter(x, filter=rep(1/3,3),method = "convolution",sides=1)
> lines(y,col="red")
> points(y,pch=8,col="red")
```

观察图 9-35, 平滑效果不错, 红色曲线及星号表示平滑后的数据, 蓝色曲线及空心圆点表示原数据。上述 R 程序通过将 filter 参数设为 rep(1/3,3), 使每个时间点的权值均为 1/3, 且前期数据取前 3 天的数据, 从而生成当天的平滑数据值。

## 2. 加权移动平均

加权移动平均给固定跨越期限内的每个变量值以不相等的权重, 其原理是: 历史各期产品需求的数据信息对预测未来期内的需求量的作用是不一样的, 除了以  $n$  为周期的周期性变化外, 远离目标期的变量值的影响力相对较低, 故应给予较低的权重。简单的移动平均法实质上也是加权移动平均, 只不过它是各元素的权重都相等。

加权移动平均的计算公式如下:

$$y[t-1]=a_1\times x[t-1]+a_2\times x[t-2]+a_3\times x[t-3]+\cdots+a_n\times x[t-n]$$

上式中,  $a$  为权值,  $x$  为数据,  $y$  为预测值,  $n$  为参与计算的数据期数, 且权值之和为 1。

在 R 语言中, 可调用 filter 函数进行线性过滤, 实现加权移动平均算法, 使用时需要注意以下事项:

- ❑ filter 参数 (该参数指定了参加平均计算的前期数据的权重) 设定为参与计算时间点的各自权重。
- ❑ method 参数指定为 convolution, 表示使用移动平均法, 并将 sides 参数设置为 1, 表示使用单边卷积。
- ❑ filter 函数对数据进行预测时, 参与计算的前期数据不但包括前  $n-1$  期的数据, 还包括当前数据, 预测值的计算公式如下:

$$y[t]=f[1]\times x[t]+f[2]\times x[t-1]+f[3]\times x[t-2]+\cdots+f[n-1]\times x[t-(n-1)]$$

以上公式中,  $t$  为时间,  $f$  为该时间点数据的权重值。

以表 9-4 所示的 X 商品的销量为例, 设当前时间点  $t$  为 5, 期数  $n$  为 3, 当前时间点权重为 0.65, 前 1 个时间点的权重为 0.25, 前 2 个时间点的权重为 0.1,  $x$  为商品销量数组, 提取前  $n$  期的数据, 预测当前时间点的平滑值, 其计算方式如下:

$$y[5]=x[5]\times 0.65+x[4]\times 0.25+x[3]\times 0.1=121\times 0.65+176\times 0.25+54\times 0.1=128.05$$

下面编写 R 代码, 对表 9-4 所示的销量数据进行平滑, 效果如图 9-36 所示。

```
>
x<-c(12,123,54,176,121,134,198,155,122,111,133,244,278,232,267,222,187,193,245,110,132)
> plot(x,type="o",col="blue",xlab="天",ylab="销量")
> y= filter(x, filter=c(0.65,0.25,0.1),method = "convolution",sides=1)
> lines(y,col="red")
> points(y,pch=8,col="red")
```

观察图 9-36, 平滑效果不错, 红色曲线及星号表示平滑后的数据, 蓝色曲线以及空心圆点表示原数据。上述 R 程序通过将 filter 参数设为 c(0.65,0.25,0.1), 将从预测时间点开始依次向前的 3 个时间点的权值分别设置为 0.65、0.25、0.1, 从而生成当天的平滑数据值。

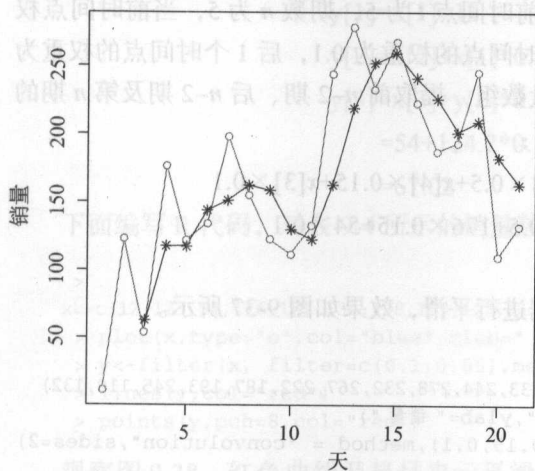


图 9-35 简单移动平均

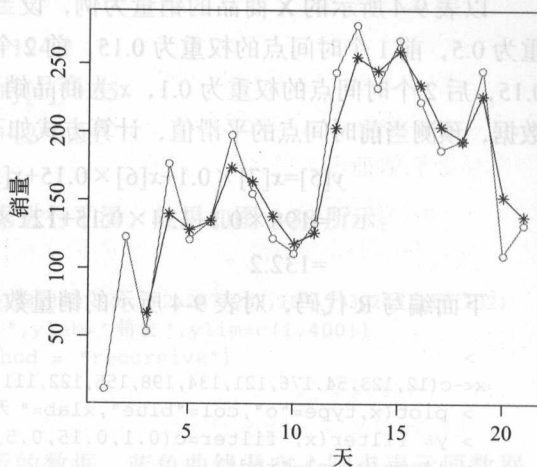


图 9-36 加权移动平均

此外, 还可输出 y 值, 从以下输出结果可以看出, 第 5 个时间点的平滑值是 128.05, 与前面手工计算的一致。

```
> y
Time Series:
Start = 1
End = 21
Frequency = 1
[1] NA NA 67.05 140.20 128.05 134.95 174.30 163.65 137.85 118.15
[11] 126.40 202.95 255.00 244.70 259.35 234.25 203.75 194.40 226.20 152.05
[21] 137.80
```

### 3. 双边卷积

双边卷积算法与简单移动平均法、加权移动平均法相似, 不同之处在于其计算平均值的依据不仅包括以前时间点的数据, 而且还包括以后时间点的数据。

在 R 语言中, 可调用 filter 函数进行线性过滤, 实现双边卷积算法, 使用时需要注意以下事项:



□ **filter** 参数（该参数指定了参加平均计算的前期数据的权重）设定为参与计算时间点各自的权重。

□ **method** 参数指定为 **convolution**，表示使用移动平均法，并将 **sides** 参数设置为 2，表示使用双边卷积。

□ **filter** 函数对数据进行预测时，参与计算的数据不但包括前  $n-1$  期的数据和当前数据，还包括以后时间点的数据。以期数是 5 为例，计算当前时间点  $t$  的预测值  $y[t]$ ，公式如下：

$$y[t]=f[1] \times x[t+2]+f[2] \times x[t+1]+f[3] \times x[t]+f[4] \times x[t-1]+f[5] \times x[t-2]$$

以上公式中， $t$  为时间， $f$  为该时间点数据的权重值。

以表 9-4 所示的 X 商品的销量为例，设当前时间点  $t$  为 5，期数  $n$  为 5，当前时间点权重为 0.5，前 1 个时间点的权重为 0.15，前 2 个时间点的权重为 0.1，后 1 个时间点的权重为 0.15，后 2 个时间点的权重为 0.1， $x$  为商品销量数组，提取前  $n-2$  期、后  $n-2$  期及第  $n$  期的数据，预测当前时间点的平滑值，计算方式如下：

$$\begin{aligned} y[5] &= x[7] \times 0.1 + x[6] \times 0.15 + x[5] \times 0.5 + x[4] \times 0.15 + x[3] \times 0.1 \\ &= 198 \times 0.1 + 134 \times 0.15 + 121 \times 0.5 + 176 \times 0.15 + 54 \times 0.1 \\ &= 132.2 \end{aligned}$$

下面编写 R 代码，对表 9-4 所示的销量数据进行平滑，效果如图 9-37 所示。

```
>
x<-c(12,123,54,176,121,134,198,155,122,111,133,244,278,232,267,222,187,193,245,110,132)
> plot(x,type="o",col="blue",xlab="天",ylab="销量")
> y= filter(x, filter=c(0.1,0.15,0.5,0.15,0.1),method = "convolution",sides=2)
> lines(y,col="red")
> points(y,pch=8,col="red")
```

观察图 9-37，平滑效果不错，红色曲线及星号表示平滑后的数据，蓝色曲线及空心圆点表示原数据。上述 R 程序通过将 **filter** 参数设为 **c(0.1,0.15,0.5,0.15,0.1)**，将从预测时间点后 2 个时间点、当前时间点及之前 2 个时间点的权值分别设置为 0.1、0.15、0.5、0.15、0.1，从而生成当天的平滑数据值。

此外，还可输出  $y$  值，从以下输出结果可以看出，第 5 个时间点的平滑值是 132.20，与前面手工计算的一致。

```
> c(y)
[1] NA NA 85.15 139.95 132.20 147.95 166.65 150.00 134.00 133.65
159.75 217.95 250.40 244.35 248.10 221.60 206.95
[18] 194.50 199.85 NA NA
```

### 9.3.3 递归线性过滤

递归线性过滤相当于自回归法，在 R 语言中，可调用 **filter** 函数进行递归线性过滤，使用时需要注意以下事项：

□ **filter** 参数（该参数指定了参加平均计算的前期数据的权重）设定为参与计算时间点各



自的权重。

□ **method** 参数指定为 **recursive**，表示使用递归线性过滤。

□ **filter** 函数对数据进行预测时，参与计算的数据包括当前时间点的数据和以前  $n-1$  个时间点的预测值。当前时间点  $t$  的预测值  $y[t]$  的计算公式如下：

$$y[t] = x[t] + f[1] \times y[t-1] + \dots + f[n-1] \times y[t-(n-1)]$$

以上公式中， $t$  为时间， $f$  为该时间点数据的权重值。

以表 9-4 所示的 X 商品的销量为例，设当前时间点  $t$  为 3，期数  $n$  为 2，前 1 个时间点的权重为 0.1，前 2 个时间点的权重为 0.05， $x$  为商品销量数组，提取前  $n-2$  期预测值  $y$  及第  $n$  期值  $x$ ，预测当前时间点的平滑值，计算方式如下：

$$y[1] = x[1] = 12$$

$$y[2] = x[2] + x[1] \times 0.1 = 124.2$$

$$y[3] = x[3] + y[2] \times 0.1 + y[1] \times 0.05$$

$$= 54 + 124.2 \times 0.1 + 12 \times 0.05$$

$$= 67.02$$

下面编写 R 代码，对表 9-4 所示的销量数据进行平滑，效果如图 9-38 所示。

```
>
x<-c(12,123,54,176,121,134,198,155,122,111,133,244,278,232,267,222,187,193,245,110,132)
> plot(x,type="o",col="blue",xlab="天",ylab="销量",ylim=c(1,400))
> y<-filter(x, filter=c(0.1,0.05),method = "recursive")
> lines(y,col="red")
> points(y,pch=8,col="red")
```

观察图 9-38，红色曲线及星号表示平滑后的数据，蓝色曲线及空心圆点表示原数据。上述 R 程序通过将 **filter** 参数设为 **c(0.1,0.05)**，将以前 2 个时间点的权值分别设置为 0.1 和 0.05，从而生成当天的平滑数据值。

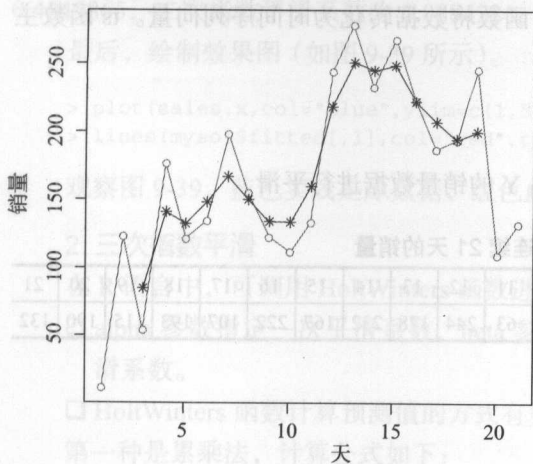


图 9-37 双边卷积

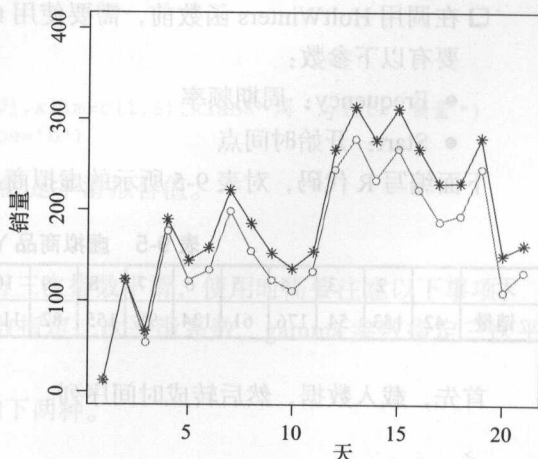


图 9-38 递归线性过滤

此外，还可输出  $y$  值，从以下输出结果可以看出，第 3 个时间点的平滑值是 67.02，与前面手工计算的一致。

```
> c(y)
[1] 12.0000 124.2000 67.0200 188.9120 143.2422 157.7698 220.9391 184.9824
[9] 151.5452 135.4036 154.1176 266.1819 312.3241 276.5415 310.2704 266.8541
[17] 229.1989 229.2626 279.3862 149.4018 160.9095
```

9.3.4 指数平滑

指数平滑 (Exponential Smoothing, ES) 法是布朗 (Robert G.Brown) 所提出的，布朗认为时间序列的态势具有稳定性或规则性，所以时间序列可被合理地顺势推延，最近的过去数据趋势，在某种程度上会持续到未来，因此将较大的权重放在最近的资料上。指数平滑法是在移动平均法的基础上发展起来的一种时间序列分析预测法，它是通过计算指数平滑值，配合一定的时间序列预测模型对现象的未来进行预测，其原理是任一期的指数平滑值都是本期实际观察值与前一期指数平滑值的加权平均。

根据平滑次数不同，指数平滑法分为：一次指数平滑法、二次指数平滑法和三次指数平滑法等。一次指数平滑法是根据前期的实测数和预测数，以一次平滑系数为权重，进行加权平均，从而预测未来时间趋势的方法；二次指数平滑法是对一次指数平滑的再平滑，它保留并更新了平滑后的信号及平滑后的趋势，它增加了二次趋势平滑系数；三次指数平滑法是在二次平滑的基础上再添加第三个量，用来描述周期性，它是二次平滑的再平滑，增加了三次周期平滑系数。

1. 二次指数平滑

在 R 语言中，可调用 HoltWinters 函数进行二次指数平滑，使用时需要注意以下事项：

- ❑ alpha 参数指定一次平滑系数，beta 参数指定二次平滑系数，gamma 参数设置为 FALSE 表示不进行三次平滑，如果不指定平滑系数，HoltWinters 函数将自动计算最优系数。
- ❑ 在调用 HoltWinters 函数前，需要使用 ts 函数将数据转化为时间序列向量。ts 函数主要有以下参数：
  - Frequency：周期频率
  - Start：开始时间点

下面编写 R 代码，对表 9-5 所示的虚拟商品 Y 的销量数据进行平滑。

表 9-5 虚拟商品 Y 连续 21 天的销量

天	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
销量	42	153	54	176	61	134	98	155	82	111	63	244	178	232	167	222	107	193	115	190	132

首先，载入数据，然后转成时间序列。

```
>
x<-c(42,153,54,176,61,134,98,155,82,111,63,244,178,232,167,222,107,193,115,190,132)
```

```
> ts(x,frequency=7,start=c(1,1))->sales.x
> sales.x
Time Series:
Start = c(1, 1)
End = c(3, 7)
Frequency = 7
[1] 42 153 54 176 61 134 98 155 82 111 63 244 178 232 167 222 107 193
[19] 115 190 132
```

观察 sales.x 的输出结果, 表 9-5 所示为 21 天的销售数据, 指定周期为 7 天后, 即形成以周为周期的时间序列, Start 表示数据的周期起始为 c(1,1), 即: 第 1 周的第 1 天, End 表示数据的周期终止为 c(3,7), 即: 第 3 周的第 7 天。

然后, 进行二次平滑。

```
> HoltWinters(sales.x, gamma=FALSE)->mysol
> mysol
Holt-Winters exponential smoothing with trend and without seasonal component.
```

Call:

```
HoltWinters(x = sales.x, gamma = FALSE)
```

Smoothing parameters:

alpha: 0.4649865

beta : 0.8894224

gamma: FALSE

Coefficients:

[,1]

a 143.102068

b 2.827244

>

观察 mysol 输出, HoltWinters 函数计算出最佳平滑系数, 其中, 一次指数平滑系数为 0.4649865, 二次指数平滑系数为 0.8894224。

最后, 绘制效果图 (如图 9-39 所示)。

```
> plot(sales.x,col="blue",ylim=c(1,500),xlim=c(1,5),xlab="周",ylab="销量")
> lines(mysol$fitted[,1],col="red",type="b")
```

观察图 9-39, 蓝色实线是原数据, 红色虚线是平滑拟合值。

## 2. 三次指数平滑

在 R 语言中, 可调用 HoltWinters 函数进行三次指数平滑, 使用时需要注意以下事项:

□ alpha 参数指定一次平滑系数, beta 参数指定二次平滑系数, gamma 参数指定三次平滑系数。

□ HoltWinters 函数计算预测值的方式有如下两种。

第一种是累乘法, 计算公式如下:

$$a[t]=\alpha(Y[t]/s[t-p])+(1-\alpha)(a[t-1]+b[t-1])$$

$$\begin{aligned}b[t]&=\beta(a[t]-a[t-1])+(1-\beta)b[t-1] \\s[t]&=\gamma(Y[t]/a[t])+(1-\gamma)s[t-p] \\Yhat[t+h]&=(a[t]+h \times b[t]) \times s[t-p+1+(h-1) \bmod p]\end{aligned}$$

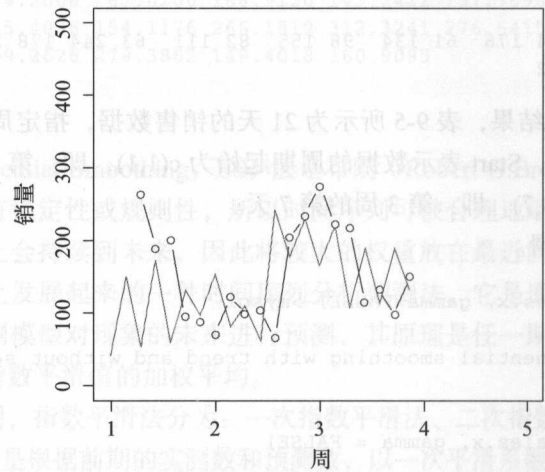


图 9-39 二次指数平滑

其中， $\alpha$ 、 $\beta$ 、 $\gamma$  分别为一次、二次、三次平滑系数，Yhat 为预测值。

第二种是累加法，计算公式如下：

$$\begin{aligned}a[t]&=\alpha(Y[t]-s[t-p])+(1-\alpha)(a[t-1]+b[t-1]) \\b[t]&=\beta(a[t]-a[t-1])+(1-\beta)b[t-1] \\s[t]&=\gamma(Y[t]-a[t])+(1-\gamma)s[t-p] \\Yhat[t+h]&=(a[t]+h \times b[t]) \times s[t-p+1+(h-1) \bmod p]\end{aligned}$$

由 seasonal 参数指定三次平滑的方式，分别为 additive（累加方式）、multiplicative（累乘方式），默认为 additvie。

□ 调用 HoltWinters 函数前，需要使用 ts 函数将数据转化为时间序列向量。ts 函数主要有以下参数：

- Frequency：周期频率
- Start：开始时间点

下面编写 R 代码，对表 9-4 所示的虚拟商品 X 的销量数据进行平滑。

首先，载入数据，然后转成时间序列。

```
>
x<-c(12,123,54,176,121,134,198,155,122,111,133,244,278,232,267,222,187,193,245,110,132)
> ts(x,frequency=7,start=c(1,1))>sales.x
> sales.x
Time Series:
Start = c(1, 1)
End = c(3, 7)
Frequency = 7
```

```
[1] 12 123 54 176 121 134 198 155 122 111 133 244 278 232 267 222
[17] 187 193 245 110 132
```

观察 sales.x 的输出结果, 表 9-4 所示为 21 天的销售数据, 指定周期为 7 天后, 即形成以周为周期的时间序列, Start 表示数据的周期起始为 c(1,1), 即: 第 1 周的第 1 天, End 表示数据的周期终止为 c(3,7), 即: 第 3 周的第 7 天。

然后, 进行三次平滑。

```
> HoltWinters(sales.x)->mysol
> mysol
Holt-Winters exponential smoothing with trend and additive seasonal component.

Call:
HoltWinters(x = sales.x)

Smoothing parameters:
alpha: 0.6264084
beta : 0
gamma: 1

Coefficients:
      [,1]
a 122.270966
b   8.447279
s1 36.581455
s2 -20.578516
s3 -57.187646
s4 -26.947654
s5 24.695147
s6 -30.083230
s7  9.729034
```

观察 mysol 输出, HoltWinters 函数计算出最佳平滑系数, 其中, 一次指数平滑系数为 0.6264084, 二次指数平滑系数为 0, 三次指数平滑系数为 1。此外, 商品 X 的销量周期系数为 s1 到 s7。

再绘制效果图, 如图 9-40 所示。

```
> plot(sales.x,col="blue",ylim=c(1,500),xlim=c(1,5),xlab="周",ylab="销量")
> lines(mysol$fitted[,1],col="red",type="b")
```

观察图 9-40, 蓝色实线是原数据, 红色虚线是平滑拟合值。

最后, 对以后的趋势进行预测。表 9-4 所示的数据是虚拟商品 X21 天的销量, 现在对以后 14 天的销量进行预测, 预测效果如图 9-41 所示。

```
> predict(mysol, 14, prediction.interval = TRUE)->p
> plot(mysol,p,xlab="周",ylab="销量",xlim=c(1,7),main="销量预测")
```

观察图 9-41, 黑色线为原数据, 红色线为平滑拟合值。虚线右侧的部分是以后 14 天的虚拟商品 X 的预测销量部分, 该部分的中间红线为预测走势曲线, 上方和下方曲线分别为预



测量置信区间的上限和下限。

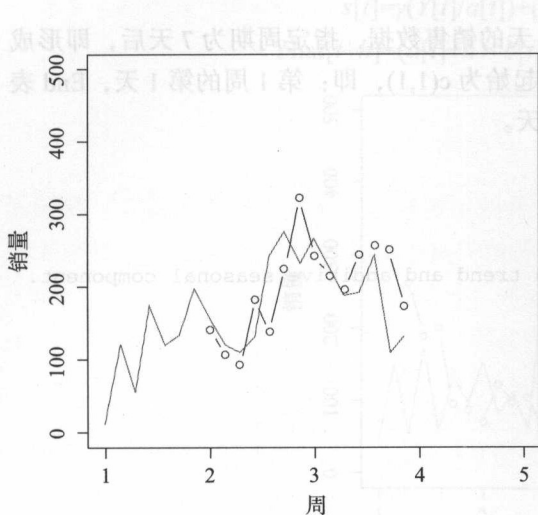


图 9-40 三次指数平滑

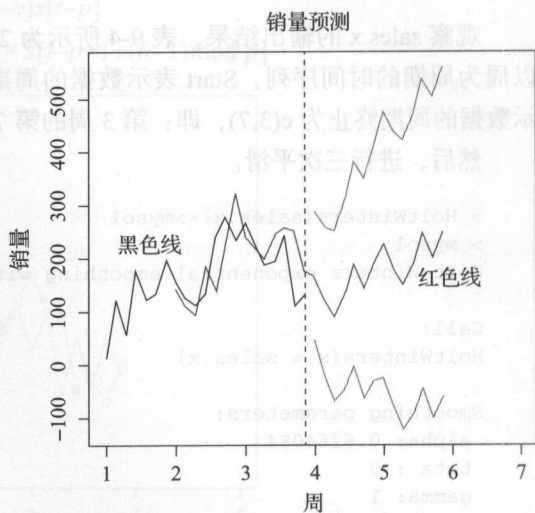


图 9-41 三次指数平滑预测

## 9.4 小结

本章首先介绍了数据拟合的技术。数据拟合是通过理想的假设方程来拟合数据的，得到这个假设方程主要有两种方式：第一，通过观察样本数据点的分布来估计所属函数的图像，在拟合后计算相关统计指标，评估拟合的效果；第二，以假设方程的自变量为输入样本，以变量为样本的目标输出，通过神经网络进行训练，以权值矩阵和众多神经元的激活函数等神经网络组件完成输入到输出的映射，其实质是建立一个更复杂的拟合模型来实现数据拟合。

然后讲解了移动平均法、递归线性过滤法、指数平滑法等平滑方法。数据或曲线平滑通过建立近似函数发现数据中的主要模式，去除噪声、结构细节或瞬时现象，减少不必要的波动，从而实现数据集的平滑。

## 思考题

(1) 假设有两类数据，下面是这两类数据的分布散点图，从图像上分析拟合这些数据的函数方程。

(2) 编写 Python 代码实现多层感知器，拟合  $y=0.7\sin(x)+0.3\cos(x)$  函数。

(3) 本章讲解了通过线性滤波去除背景音乐的方法。请尝试用非线性滤波解决这个问题，即：使用多层感知器对背景音乐进行拟合，然后将它从语音文件中去除。



## 图像算法案例

图 9-40 三次样条平滑

机器学习算法可对数字图像进行加工和处理,以便进一步进行特征提取的工作,主要目标为从图像中挖掘所需要的知识和信息,主要解决图像锐化、图像除噪、图像增强、图像分类、图像识别等问题,其算法过程主要包括:图像获取、预处理、特征提取、加工分析、提取知识等。

### 10.1 图像边缘算法

#### 10.1.1 数字图像基础

再次复习一下数字图像的知识。数字图像在计算机中保存为二维整数数组,数组中元素是二维图像中的像素,每个像素都用有限数值表示,对应于二维空间中一个特定的位置。图像是由二维像素点组成的矩阵,通常每个像素点由 3 个元素组成——红、绿、蓝,这 3 个基本分量可以组成高清的图像。也可以把图像上的每个像素点理解为  $(x,y,z)$  这样的点,这个点定义在三维空间,每一维分别代表红、绿、蓝分量。

假设将像素的顺序定义为:蓝、绿、红,那么就可以定义一个像素点为  $(blue, green, red)$ 。每个图像由大量的像素点组成,如果将这个像素点组成的矩阵定义为  $H \times W$  大小 ( $H$  为高,  $W$  为宽),那么就得到了一个  $H \times W \times 3$  的矩阵 (“3” 表示像素点的数值由 3 个基本分量组成)。OpenCV 作为图像算法库,对图像矩阵也是这么定义的。请提前安装好 OpenCV,本章大部分例子都需要它的 Python 绑定库。

假设图像矩阵的变量名为 `img`,现在要用 Python 实现一个蓝色分量为 200、绿色为 100、红色为 50 的像素点的定义,这个像素点位于图像的  $300 \times 150$  处。代码如下:

```
img[300,150,0]=200
img[300,150,1]=100
img[300,150,2]=50
```

### 10.1.2 算法描述

算法的基本原理是：将当前像素与邻接的下部和右部的像素进行比较，如果相似，则将当前像素设置为白色，否则设置为黑色。如何判定像素相似呢？应用欧氏距离算法，将一个像素的 3 个色彩分量映射在三维空间中，如果 2 个像素点的欧氏距离小于某个常数的阈值，就认为它们相似。算法的最终效果如图 10-1 所示。图 10-1 中左图是原图像，右图是计算图像边缘的结果。

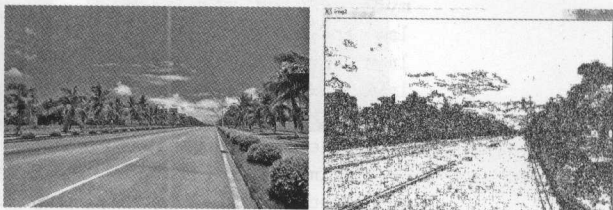


图 10-1 图像边缘

上面涉及的算法其关键是欧氏距离计算。下面用 Python 编写计算欧氏距离的函数。

```
def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))
```

完整的代码为：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-1.py
import cv2
import numpy as np

fn="test1.jpg"
def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

if __name__ == '__main__':

    print 'loading %s ...' % fn
    print 'working',
    myimg1 = cv2.imread(fn)
    w=myimg1.shape[1]
    h=myimg1.shape[0]
    sz1=w
    sz0=h
    # 创建空白图像
    myimg2=np.zeros((sz0,sz1,3), np.uint8)
    # 对比产生线条
```

```

black=np.array([0,0,0])
white=np.array([255,255,255])
centercolor=np.array([125,125,125])
for y in xrange(0,sz0-1):
    for x in xrange(0,sz1-1):

```

```

        mydown=myimg1[y+1,x,:]
        myright=myimg1[y,x+1,:]

        myhere=myimg1[y,x,:]
        lmyhere=myhere
        lmyright=myright
        lmydown=mydown
        if get_EuclideanDistance(lmyhere,lmydown)>16 and get_EuclideanDistance(lmyhere,lmyright)>16:
            myimg2[y,x,:]=black
        elif get_EuclideanDistance(lmyhere,lmydown)<=16 and get_EuclideanDistance(lmyhere,lmyright)<=16:
            myimg2[y,x,:]=white
        else:
            myimg2[y,x,:]=centercolor
        print '.',
cv2.namedWindow('img2')
cv2.imshow('img2', myimg2)
cv2.waitKey()
cv2.destroyAllWindows()

```

## 10.2 图像匹配

图像匹配算法是基于像素的比较和计算来实现的方法。如图 10-2 所示是美国的 X-47B，它是人类历史上第一架无需人工干预、完全由计算机智能操纵的无人驾驶飞机。现在以其在航母起飞的图像为例讲解本节内容。



图 10-2 X-47B 航母起飞



如图 10-3 所示为图 10-2 的两张局部切片图。我们的任务是找到两张切片图在图 10-2 中的位置，并将它们标注出来。

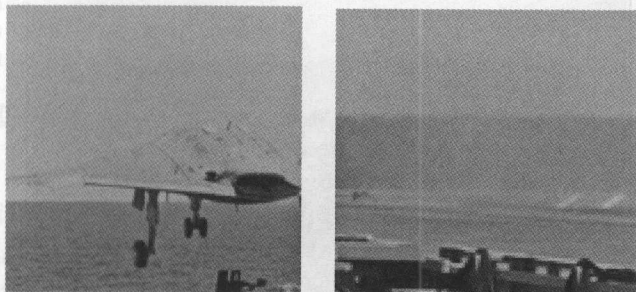


图 10-3 两张 X-47B 起飞画面切片图

### 10.2.1 差分矩阵求和

差分算法的核心在于差分矩阵，实质为差异矩阵，计算公式很简单：

差分矩阵 = 图像 A 矩阵数据 - 图像 B 矩阵数据

算法过程是：首先，计算两个图像的矩阵数据之间差异分析图像的相似性；然后，设置一个阈值进行比较，如果差分矩阵的所有元素之和在阈值以内，则表示这两张图像是相似的，且描述了同一物体。另外，它要求两个图像的大小相同，大小处理对于计算机来说不成问题，改变图像尺寸的算法已非常成熟，实现起来很方便。

编写程序实现这个算法的基本思路为：将图 10-3 所示的切片图在图 10-2 中进行移动，并计算两个图像的差分矩阵，如果差分矩阵的所有元素之和小于 1，则认为找到了切片图在图像中的位置。实现该算法的 Python 代码如下：

```
def showpiclocation(img, findimg):
    # 定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]
    fh=findimg.shape[0]
    findpt=None
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            comp_tz=img[now_h:now_h+fh,now_w:now_w+fw,:]-findimg
            if np.sum(comp_tz)<1:
                findpt=now_w,now_h
        print ".",
    if findpt!=None:
        cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh), (255,0,0))
    return img
```

如图 10-4 所示为算法效果图，看上去识别效果不错。

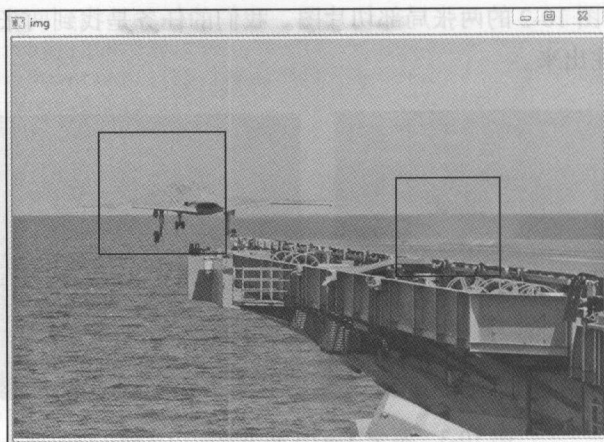


图 10-4 切片识别效果图

完整的 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-2.py
# 简单定位图像

import cv2
import numpy as np
print 'loading ...'
def showpiclocation(img,finding):
    # 定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=finding.shape[1]
    fh=finding.shape[0]
    findpt=None
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            comp_tz=img[now_h:now_h+fh,now_w:now_w+fw,:]-finding
            if np.sum(comp_tz)<1:
                findpt=now_w,now_h
        print ".,",
    if findpt!=None:
        cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh),(255,0,0))
    return img

fn='pictest.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)
```

```

myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

## 10.2.2 差分矩阵均值

刚才小试牛刀，通过对差分矩阵中所有元素求和完成了匹配，效果不错。当数字图像质量较差时，则需要计算差分矩阵的均值，并为均值设一个适当的阈值。

下面仍用 Python 编写算法，在目标图中加上少量（50000 个）不同颜色的噪声点，从而测试算法在弱噪声环境下的有效性。因为图像质量不高，存在很多噪声点，所以要将差分矩阵均值的阈值设置得稍大一点，这里设为 20。通常来说，阈值为 10 ~ 200，阈值越大，能容忍的噪声点越多。但如果阈值超过 200，最好使用下一节介绍的欧氏距离算法。如图 10-5 所示是算法应用效果。

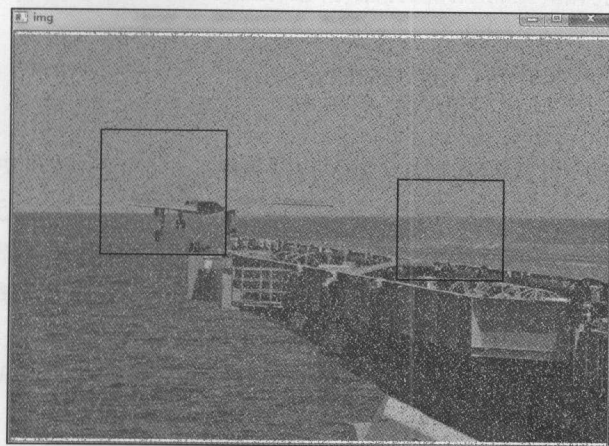


图 10-5 弱噪声切片识别效果图（附彩图）

Python 代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-3.py
# 少量噪声定位图像

```

```

import cv2
import numpy as np

```

```

print 'loading ...'

```

```

def showpiclocation(img,finding):

```

```

    # 定位图像

```

```

    w=img.shape[1]

```

```

h=img.shape[0]
fw=findimg.shape[1]
fh=findimg.shape[0]
findpt=None
for now_h in xrange(0,h-fh):
    for now_w in xrange(0,w-fw):
        comp_tz=img[now_h:now_h+fh,now_w:now_w+fw,:]-findimg
        if abs(np.mean(comp_tz))<20:
            findpt=now_w,now_h
            print "ok"
    print ".",
if findpt!=None:
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh),(0,0,255))
return img

def addnoise(img):
    counn=50000
    for k in xrange(0,counn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        img[xj,xi,0]= 255 *np.random.rand()
        img[xj,xi,1]= 255 *np.random.rand()
        img[xj,xi,2]= 255 *np.random.rand()

fn='pictest.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)
addnoise(myimg)
myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

可见，在弱噪声的情况下，差分算法仍然具有很好的匹配效果。

### 10.2.3 欧氏距离匹配

#### 1. 强噪声图像匹配

对强噪声环境下的图像进行匹配时，欧氏距离匹配方法相对于以上两种方法会有更好的效果。

仍以上面的图像为目标进行讲解。首先，在目标图像中加上更多的（500000个）不同颜色的噪声点。代码如下：

```

def addnoise(img):
    counn=500000
    for k in xrange(0,counn):

```



```

xi = int(np.random.uniform(0,img.shape[1]))
xj = int(np.random.uniform(0,img.shape[0]))
img[xj,xi,0]= 255 *np.random.rand()
img[xj,xi,1]= 255 *np.random.rand()
img[xj,xi,2]= 255 *np.random.rand()

```

生成强噪声环境下的图像，如图 10-6 所示。

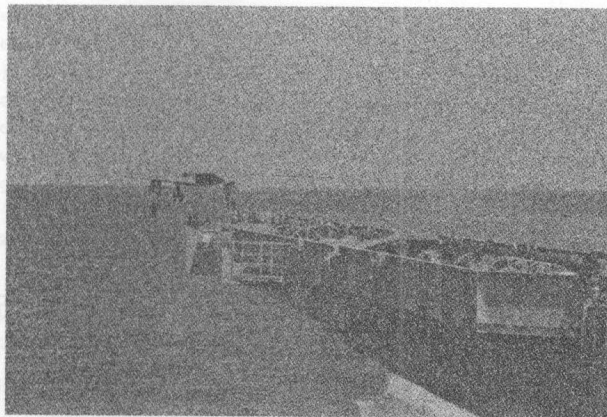


图 10-6 强噪声图像（附彩图）

现在要应用欧氏距离对如图 10-6 所示的目标图完成匹配。其核心算法为：设图像矩阵有  $n$  个元素，用  $n$  个元素值  $(x_1, x_2, \dots, x_n)$  组成该图像的特征组，特征组形成了  $n$  维空间，特征组中的特征码构成每一维的数值。在  $n$  维空间下，两个图像矩阵各形成了一个点，然后计算这两个点之间的距离，距离最小者为最匹配的图像。

如图 10-7 所示是算法应用的效果图，欧氏距离算法成功地找到了匹配位置。在图像如此模糊的情况下，人用肉眼都很难进行图像匹配。由此可见，机器学习算法在某些方面胜过人类的“智能”。

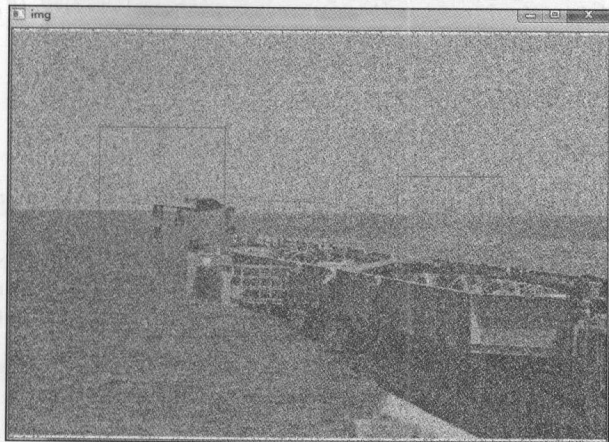


图 10-7 强噪声切片识别效果图（附彩图）



完整的 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-4.py
# 大量噪声定位图像

import cv2
import numpy as np

print 'http://blog.csdn.net/myhaspl'
print 'myhaspl@qq.com'
print
print 'loading ...'

def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

def findpic(img,finding,h,fh,w,fw):
    minds=1e8
    mincb_h=0
    mincb_w=0
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            my_img=img[now_h:now_h+fh,now_w:now_w+fw,:]
            my_finding=finding
            dis=get_EuclideanDistance(my_img,my_finding)
            if dis<minds:
                mincb_h=now_h
                mincb_w=now_w
                minds=dis
        print ".",
    findpt=mincb_w,mincb_h
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh),(0,0,255))
    return img

def showpiclocation(img,finding):
    # 定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=finding.shape[1]
    fh=finding.shape[0]
    return findpic(img,finding,h,fh,w,fw)

def addnoise(img):
    coutn=500000
    for k in xrange(0,coutn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
```

```
img[xj,xi,0]= 255 *np.random.rand()
img[xj,xi,1]= 255 *np.random.rand()
img[xj,xi,2]= 255 *np.random.rand()
```

```
fn='pictest.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)
addnoise(myimg)
myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

## 2. 变形图像匹配

欧氏距离方法不仅对强噪声图像匹配有效，而且对变形后的图像匹配效果也不错。如图 10-8 所示就是应用欧氏距离方法对有倾斜角度的图像进行匹配的效果。



图 10-8 倾斜图像匹配效果图

算法原理前面已经解说过，在此不重复。相关的 Python 代码实现如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-5.py
# 图像倾斜后定位图像
```

```
import cv2
```

```

import numpy as np

print 'loading ...'

def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

def findpic(img,findimg,h,fh,w,fw):
    minds=1e8
    mincb_h=0
    mincb_w=0
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            my_img=img[now_h:now_h+fh,now_w:now_w+fw,:]
            my_findimg=findimg
            dis=get_EuclideanDistance(my_img,my_findimg)
            if dis<minds:
                mincb_h=now_h
                mincb_w=now_w
                minds=dis
        print ".",
    findpt=mincb_w,mincb_h
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh),(0,0,255))
    return img

def showpiclocation(img,findimg):
    # 定位图像
    w=img.shape[1]
    h=img.shape[0]
    fw=findimg.shape[1]
    fh=findimg.shape[0]
    return findpic(img,findimg,h,fh,w,fw)

fn='pictestxz.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)

myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

欧氏距离对弱噪声环境下的变形图像仍有不错的效果，如图 10-9 所示。

该算法的 Python 实现代码如下：



图 10-9 弱噪声变形图像匹配

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-6.py
# 图像倾斜后加噪声点，定位图像
import cv2
import numpy as np

print 'loading ...'
def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))

def findpic(img,finding,h,fh,w,fw):
    minds=1e8
    mincb_h=0
    mincb_w=0
    for now_h in xrange(0,h-fh):
        for now_w in xrange(0,w-fw):
            my_img=img[now_h:now_h+fh,now_w:now_w+fw,:]
            my_finding=finding
            dis=get_EuclideanDistance(my_img,my_finding)
            if dis<minds:
                mincb_h=now_h
                mincb_w=now_w
                minds=dis
        print ".",
    findpt=mincb_w,mincb_h
    cv2.rectangle(img, findpt, (findpt[0]+fw,findpt[1]+fh),(0,0,255))
    return img

def showpiclocation(img,finding):
```

```

# 定位图像
w=img.shape[1]
h=img.shape[0]
fw=findimg.shape[1]
fh=findimg.shape[0]
return findpic(img, findimg, h, fh, w, fw)

def addnoise(img):
    coutn=50000
    for k in xrange(0,coutn):
        xi = int(np.random.uniform(0,img.shape[1]))
        xj = int(np.random.uniform(0,img.shape[0]))
        img[xj,xi,0]= 255 *np.random.rand()
        img[xj,xi,1]= 255 *np.random.rand()
        img[xj,xi,2]= 255 *np.random.rand()

fn='pictestxz.png'
fn1='pictestt1.png'
fn2='pictestt2.png'
myimg=cv2.imread(fn)
myimg1=cv2.imread(fn1)
myimg2=cv2.imread(fn2)
addnoise(myimg)
myimg=showpiclocation(myimg,myimg1)
myimg=showpiclocation(myimg,myimg2)
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

## 10.3 图像分类

近年来随着多媒体技术的发展,图像分类技术受到了普遍的关注,目前采用的方法以机器学习算法为主。图像分类利用计算机对图像进行分析,根据图像信息的不同特征,将不同类别的图像区分开来。

图像分类的算法过程如下:

- 1) 准备样本图像。样本图像的要求是:能代表所属类别中尽可能多的图像。
- 2) 提取每个样本的特征后,形成类别特征码。
- 3) 应用机器学习算法对类别特征码进行学习,提取特征码包含的图像知识。
- 4) 判断未知图像所属类别。

### 10.3.1 余弦相似度

余弦相似度通过测量两个向量内积空间的夹角的余弦值来度量它们之间的相似性,尤其适用于任何维度的向量比较中,因此属于高维空间应用较多的机器学习算法。通常来说,数字图像包含的特征码较多,而这些特征组就属于高维空间,这正是余弦相似度算法应用的范



围, 算法将每个图像的特征组转化为高维空间的向量, 两个向量之间的角度之余弦值可用于确定两个向量是否大致指向相同的方向。

在图像分类中应用余弦相似度算法的关键在于: 计算这些代表每个图像特征的向量的内积空间的夹角余弦值, 从而度量图像之间的相似性。对于相似性的衡量标准有以下两种:

- 为相似性设置一个阈值。在这个阈值以内的都属于同一类别图像。这种标准可以将图像划分为多种类型, 例如: 高楼不但属于城市美景, 而且属于写字楼景观。
- 选择与样本向量的余弦相似度最接近 1 的图像为该类别图像。这种标准只能将图像划分为一种类别。

### 1. 算法描述

下面针对第二种衡量标准讲解余弦相似度算法。

特征提取一直是图像处理和计算机视觉研究领域中的一个值得探讨的问题, 在计算机科学、医疗辅助诊断、军事、工业测量等众多领域都广泛采用这一技术, 尤其是在计算机视觉和模式识别的研究中。如何准确定位和提取关键特征往往是首先需要解决的问题之一, 是提高识别率等问题的重要前期准备和关键因素。目前图像特征提取算法较多, 不同的算法适应于不同的图像分析任务。

本节讲述的算法基本原理是: 把图像上的点分为不同的子集, 这些子集往往属于孤立的点、连续的曲线或者连续的区域。将这些点按区域组成子集, 提取子集的特征后, 将每个子集的特征作为图像的一个特征项来进行计算。

1) 样本特征。设图像分为  $m$  个区域, 每个区域有  $n$  个像素, 每个像素在图像矩阵中以红、绿、蓝三色来表示, 且区域特征计算方式为:

$$\text{区域特征码} = \left( \frac{\sum_{i \in n} \text{像素 } i \text{ 颜色的红色分量值}}{n}, \frac{\sum_{i \in n} \text{像素 } i \text{ 颜色的绿色分量值}}{n}, \frac{\sum_{i \in n} \text{像素 } i \text{ 颜色的蓝色分量值}}{n} \right)$$

那么, 样本特征码矩阵为  $3 \times m$ , 即共 3 行  $m$  列, 这 3 行分别代表红、绿、蓝三个分量, 每列为各分量的区域特征码。

2) 类别特征。这里为每个类别准备了 3 个样本, 类别特征的计算方式为:

$$\text{类别特征码} = \left( \frac{\sum_{i \in 3} \text{样本 } i \text{ 的红色分量特征值}}{3}, \frac{\sum_{i \in 3} \text{样本 } i \text{ 的绿色分量特征值}}{3}, \frac{\sum_{i \in 3} \text{样本 } i \text{ 的蓝色分量特征值}}{3} \right)$$

其中, 样本  $i$  是属于该类别的样本。

在计算出类别特征后, 算法对样本学习完毕。当有未知图像需要分类时, 首先计算其图像的样本特征, 然后将样本特征和类别特征映射为高维空间的向量, 最后计算这两个向量的

余弦相似度，选择余弦相似度最大的类别为未知图像对应的类别。

## 2. 算法应用

下面以风景图像分类为例说明余弦相似度的应用。为每个类别各准备3个样本图像，提取类别特征码，然后将如图10-10~图10-12所示的3个待分类图像划分到蓝天风景、树林风景、瀑布风景这3个分类中。

## 3. Python 实现

1) 将图10-10~图10-12分别从上到下、从左到右分割成若干块状区域，对每块区域的图像像素特征进行提取后，形成这3个待分类图像的特征码。下面的代码所示的函数readpic定义了分割并提取特征码的操作，以待分类图像为参数调用该函数，完成特征码计算。



图 10-10 蓝天风景



图 10-11 树林风景



图 10-12 瀑布风景

```
def readpic(fn):
    # 返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (800,600))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/w_fg
    h_interval=h/h_fg
    alltz=[]
    alltz.append([])
    alltz.append([])
    alltz.append([])
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
```

```

g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
btz=np.mean(b)
gtz=np.mean(g)
rtz=np.mean(r)
alltz[0].append(btz)
alltz[1].append(gtz)
alltz[2].append(rtz)
return alltz

```

2) 计算类别特征码。通过每个类别所有样本的区域特征的平均值，提取类别特征。代码如下：

```

# 读取图像，提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x=[]
    b_tz=np.array([0,0,0])
    g_tz=np.array([0,0,0])
    r_tz=np.array([0,0,0])
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,3):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        tmptz=readpic(fn)
        mytz+=np.array(tmptz)
    mytz/=3
    train_x.append(mytz[0].tolist()+mytz[1].tolist()+mytz[2].tolist())

```

3) 计算待分类图像的特征码与每个类别特征码之间的余弦距离，距离最大者为图像所属分类。下面的代码计算了 ptest3.png 图像与每个类别特征码的余弦相似度。

```

fn='ptest3.png'
testtz=np.array(readpic(fn))
simmt=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simmtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s 属于第 %d 类'%(fn,nowi+1)

```

完整的 Python 代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-7.py
# 余弦距离识别图像类型
import numpy as np
import cv2

```

```

print u'正在处理中'
w_fg=20
h_fg=15
picflag=3
def readpic(fn):
    # 返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (800,600))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/w_fg
    h_interval=h/h_fg
    alltz=[]
    alltz.append([])
    alltz.append([])
    alltz.append([])
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz[0].append(btz)
            alltz[1].append(gtz)
            alltz[2].append(rtz)
    return alltz

def get_cossimi(x,y):
    myx=np.array(x)
    myy=np.array(y)
    cos1=np.sum(myx*myy)
    cos21=np.sqrt(sum(myx*myx))
    cos22=np.sqrt(sum(myy*myy))
    return cos1/float(cos21*cos22)

#x和d样本初始化
train_x=[]
d=[]

# 读取图像，提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x=[]
    b_tz=np.array([0,0,0])
    g_tz=np.array([0,0,0])
    r_tz=np.array([0,0,0])
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,3):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        tmpztz=readpic(fn)
        mytz+=np.array(tmpztz)
    mytz/=3

```



```

train_x.append(mytz[0].tolist()+mytz[1].tolist()+mytz[2].tolist())

fn='ptest3.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s 属于第 %d 类'%(fn,nowi+1)

fn='ptest1.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s 属于第 %d 类'%(fn,nowi+1)

fn='ptest2.png'
testtz=np.array(readpic(fn))
simtz=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simtz)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s 属于第 %d 类'%(fn,nowi+1)

```

运行上述代码，观察下面的运行结果，可见对算法任务中列举的图像识别效果不错。

正在处理中

```

ptest3.png 属于第 3 类
ptest1.png 属于第 1 类
ptest2.png 属于第 2 类

```

本节中每个类别仅使用了3个样本，基于余弦相似度的算法在样本量较小的情况下，效果其实不是最佳的。例如：在测试图像中加入另一张测试图，要用基于余弦相似度算法将它分类，那会怎样？修改10-7.py，代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com

```



```

#10-8.py
# 余弦距离识别图像类型，有一张图像不能正确识别
import numpy as np
import cv2
print u'正在处理中'
.....
.....
fn='ptest22.png'
testtz=np.array(readpic(fn))
simez=testtz[0].tolist()+testtz[1].tolist()+testtz[2].tolist()
maxtz=0
nowi=0
for i in xrange(0,picflag):
    nowsim=get_cossimi(train_x[i],simez)
    if nowsim>maxtz:
        maxtz=nowsim
        nowi=i
print u'%s 属于第 %d 类'%(fn,nowi+1)

```

程序运行结果如下：

```

正在处理中
ptest3.png 属于第 3 类
ptest1.png 属于第 1 类
ptest2.png 属于第 2 类
ptest22.png 属于第 3 类

```

从上述结果看，图像 ptest22.png（如图 10-13 所示）被错误地分到了第三类，实际它属于第二类树林风景图像。

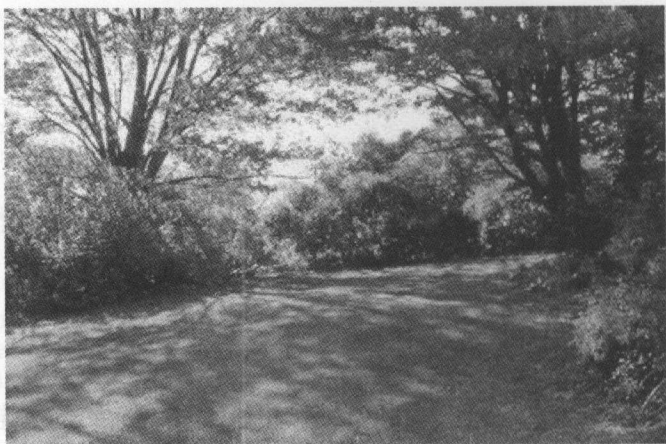


图 10-13 待分类图像

### 10.3.2 PCA 图像特征提取算法

PCA 算法基于变量协方差矩阵对信息进行压缩和处理，通常用于数据降维，可将它用于

图像矩阵降维，以降维后的矩阵为基础提取图像特征。当提取的图像特征维度比较高时，为了简化计算量以及存储空间，需要对这些高维数据进行一定程度上的降维，并尽量保证数据不失真。此外，PCA 算法还可应用于图像矩阵，它能找到变化大的维，去除掉那些变化不大的维，这样能更有效地提取图像明显特征，便于后期识别算法并进一步加工，因为图像特征组含有的不明显的特征值将会影响识别的精度。

应用 PCA 降维技术，对上节讲述的图像特征码算法进行改进，返回图像特征码。下面是用 Python 实现的基于 PCA 的图像特征码算法。

```
def readpic(fn):
    # 返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (500,400))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/20
    h_interval=h/10
    alltz=[]
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz.append([btz,gtz,rtz])
    result_alltz=np.array(alltz).T
    pca = mlpy.PCA()
    pca.learn(result_alltz)
    result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
    result_alltz =result_alltz.reshape(len(result_alltz))
    return result_alltz
```

下一节的神经网络与 SVM 图像分类算法将基于本节描述的 PCA 技术提取图像特征码，然后进一步分析和计算，得到图像所属类别。

### 10.3.3 基于神经网络的图像分类

基于神经网络的图像分类算法比余弦相似度分类算法的普适性更好，准确率更高。

#### 1. 算法描述

神经网络图像分类算法首先通过 PCA 技术提取样本图像特征码与待分类图像特征码，然后将特征码送入神经网络进行训练，让神经网络学习每个类别图像的特征，最后将未知类别图像送入神经网络，自动识别它的类型。其步骤如下：

- 1) 基于 PCA 技术提取每个样本的图像特征码。
- 2) 根据样本特征码生成输入项，根据样本所属类别生成对应的输出项。

- 3) 将输入与输出项送入非线性神经网络训练。
- 4) 基于 PCA 技术生成待分类图像的特征码。
- 5) 将待分类图像的特征码送入神经网络仿真测试, 根据神经网络输出项判断其所属类别。

## 2. 输出目标设计

神经网络的输出目标以及输出函数的设计应本着灵活实用的原则。在本例中, 神经网络的输出值为 3 个图像类别, 用数字 1 ~ 3 来表示, 但不能直接将数字作为目标输出值。常用的表示方式有以下几种。

□ 将数字转换为 1 以内的小数。比如: 乘以输出值的最大数的倒数进行调整等。

□ 按照二进制编码的思路, 将其设计为如下形式:

1:[0,0,1]

2:[0,1,0]

3:[0,1,1]

或者, 设计为以下格式 (本例采用这种格式):

1:[0,0,1]

2:[0,1,0]

3:[1,0,0]

输出函数的设计原则是: 将输出目标值转化为实际样本的输出值格式。本例中将其定义为: 神经网络输出值 = 输出目标值的最大值所在的数组索引

## 3. Python 实现

下面来看看图像特征码计算, 代码如下:

```
def readpic(fn):
    # 返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (500,400))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/20
    h_interval=h/10
    alltz=[]
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
            r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
            btz=np.mean(b)
            gtz=np.mean(g)
            rtz=np.mean(r)
            alltz.append([btz,gtz,rtz])
    result_alltz=np.array(alltz).T
    pca = mlpy.PCA()
```

```
pca.learn(result_alltz)
result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
result_alltz = result_alltz.reshape(len(result_alltz))
return result_alltz
```

接着是输入与输出项初始化。代码如下：

```
#x 和 d 样本初始化
train_x = []
d = []
sp_d = []
sp_d.append([0,0,1])
sp_d.append([0,1,0])
sp_d.append([1,0,0])
# 读取图片
for ii in xrange(1,4):
    for jj in xrange(1,4):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        pictz=readpic(fn)
        train_x.append(pictz)
        d.append(sp_d[ii-1])
myinput=np.array(train_x)
mytarget=np.array(d)
```

下面来看看训练效果。误差曲线如图 10-14 所示。

程序运行后，识别效果如下：

```
训练神经网络完毕
对样本进行测试
[1, 1, 1, 2, 2, 2, 3, 3, 3]
进行仿真
===ptest3.png===
[[ 0.96680714  0.00471284 -0.04523491]]
[3]
===ptest1.png===
[[ 0.10858063 -0.00820875  0.95785349]]
[1]
===ptest2.png===
[[ 0.01738297  0.99945777 -0.01017012]]
[2]
===ptest21.png===
[[ 0.95422417  0.00308943  0.12213834]]
[3]
===ptest22.png===
[[-0.26776152  0.99953727 -0.12122857]]
[2]
```

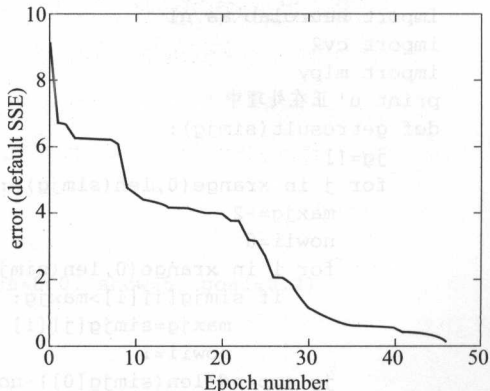


图 10-14 误差曲线

最后几行表明，无法被余弦相似度正确分类的 ptest22.png 被神经网络分类成功，但 ptest21.png（如图 10-15 所示）被错误分类。因为神经网络与 SVM 的不同之处在于，神经网络训练需要更多的样本进行训练（本例仅使用 3 个样本），否则不一定能取得更好的识别效果。

完整的 Python 代码如下：



图 10-15 错误分类的图像

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-9.py
#PCA 加上人工神经网络识别图像类型
import numpy as np
import pylab as pl
import neurolab as nl
import cv2
import mlpy
print u'正在处理中'
def getresult(simjg):
    jg=[]
    for j in xrange(0,len(simjg)):
        maxjg=-2
        nowii=0
        for i in xrange(0,len(simjg[0])):
            if simjg[j][i]>maxjg:
                maxjg=simjg[j][i]
                nowii=i
        jg.append(len(simjg[0])-nowii)
    return jg
def readpic(fn):
    # 返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (500,400))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/20
    h_interval=h/10
    alltz=[]
    for now_h in xrange(0,h,h_interval):
        for now_w in xrange(0,w,w_interval):
            b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
            g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
```



```

        r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
        btz=np.mean(b)
        gtz=np.mean(g)
        rtz=np.mean(r)
        alltz.append([btz,gtz,rtz])
    result_alltz=np.array(alltz).T
    pca = mlpy.PCA()
    pca.learn(result_alltz)
    result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
    result_alltz =result_alltz.reshape(len(result_alltz))
    return result_alltz

#x 和 d 样本初始化
train_x = []
d=[]
sp_d=[]
sp_d.append([0,0,1])
sp_d.append([0,1,0])
sp_d.append([1,0,0])
# 读取图像
for ii in xrange(1,4):
    for jj in xrange(1,4):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        pictz=readpic(fn)
        train_x.append(pictz)
        d.append(sp_d[ii-1])
myinput=np.array(train_x)
mytarget=np.array(d)
mymax=np.max(myinput)
netminmax=[]
for i in xrange(0,len(myinput[0])):
    netminmax.append([0,mymax])

print u'\n 正在建立神经网络 '
bpnet = nl.net.newff(netminmax, [5, 3])

print u'\n 训练神经网络中 ...'
err = bpnet.train(myinput, mytarget, epochs=800, show=5, goal=0.2)
if err[len(err)-1]>0.4:
    print u'\n 训练神经网络失败 ... \n'
else:
    print u'\n 训练神经网络完毕 '
    pl.subplot(111)
    pl.plot(err)
    pl.xlabel('Epoch number')
    pl.ylabel('error (default SSE)')
    print u" 对样本进行测试 "
    simd= bpnet.sim(myinput)
    mysimd=getresult(simd)
    print mysimd
    print u" 进行仿真 "
    testpictz=np.array([readpic('ptest3.png')])
    simtest=bpnet.sim(testpictz)
    mysimtest=getresult(simtest)
    print "===ptest3.png==="
    print simtest

```

```

print mysimtest
testpictz=np.array([readpic('ptest1.png')])
simtest=bpnet.sim(testpictz)
mysimtest=getresult(simtest)
print "===ptest1.png==="
print simtest
print mysimtest
testpictz=np.array([readpic('ptest2.png')])
simtest=bpnet.sim(testpictz)
mysimtest=getresult(simtest)
print "===ptest2.png==="
print simtest
print mysimtest
testpictz=np.array([readpic('ptest21.png')])
simtest=bpnet.sim(testpictz)
mysimtest=getresult(simtest)
print "===ptest21.png==="
print simtest
print mysimtest
testpictz=np.array([readpic('ptest22.png')])
simtest=bpnet.sim(testpictz)
mysimtest=getresult(simtest)
print "===ptest22.png==="
print simtest
print mysimtest
pl.show()

```

### 10.3.4 基于 SVM 的图像分类

在样本量少的情况下，SVM 分类的效果是最佳的。SVM 算法相比神经网络的优势在于：只需要少量样本，就能达到较高的识别精度。

#### 1. 算法描述

SVM 图像分类算法首先通过 PCA 技术提取样本图像特征码与待分类图像特征码，然后将特征码送入 SVM 进行训练，学习每个类别图像的特征，最后将未知类别图像送入 SVM 仿真测试，自动识别它的类型。步骤如下：

- 1) 基于 PCA 技术提取每个样本的图像特征码。
  - 2) 根据样本特征码生成输入项，根据样本所属类别生成对应的输出项。
  - 3) 将输入与输出项送入 SVM 训练。
  - 4) 基于 PCA 技术生成待分类图像的特征码。
  - 5) 将待分类图像的特征码送入 SVM 仿真测试，根据 SVM 输出项判断其所属类别。
- 其中，SVM 的输出目标可以直接使用类别序号（在本例中为数字 1 ~ 3）。

#### 2. Python 实现

1) 输入与输出项的初始化。代码如下：

```

#x 和 d 样本初始化
train_x =[]

```

```
d=[]
# 读取图像, 提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x=[]
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,4):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        tmpztz=readpic(fn)
        train_x.append(tmpztz.tolist())
    d.append(ii)
```

## 2) SVM 训练与仿真训练。代码如下:

```
x=np.array(train_x)
y=np.array(d)
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly',gamma=50).
svm.learn(x, y)
print svm.pred(x)
```

## 3) 识别效果。如下所示:

```
正在处理中
[ 1.  1.  1.  2.  2.  2.  3.  3.  3.]
ptest3.png 属于第 3 类
ptest1.png 属于第 1 类
ptest2.png 属于第 2 类
ptest21.png 属于第 2 类
ptest22.png 属于第 2 类
```

相对神经网络而言, 在每个类别仅有 3 个样本的情况下, 所有的测试图像得到了正确的分类。

以下是完整的代码:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#10-10.py
#PCA 加上 SVM 识别图像类型
import numpy as np
import cv2
import mlpy
print u'正在处理中'
w_fg=10
h_fg=5
picflag=3
def readpic(fn):
    # 返回图像特征码
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg, (400,200))
    w=img.shape[1]
    h=img.shape[0]
    w_interval=w/w_fg
    h_interval=h/h_fg
    alltz=[]
```

```

for now_h in xrange(0,h,h_interval):
    for now_w in xrange(0,w,w_interval):
        b = img[now_h:now_h+h_interval,now_w:now_w+w_interval,0]
        g = img[now_h:now_h+h_interval,now_w:now_w+w_interval,1]
        r = img[now_h:now_h+h_interval,now_w:now_w+w_interval,2]
        btz=np.mean(b)
        gtz=np.mean(g)
        rtz=np.mean(r)
        alltz.append([btz,gtz,rtz])
result_alltz=np.array(alltz).T
pca = mlpy.PCA()
pca.learn(result_alltz)
result_alltz = pca.transform(result_alltz, k=len(result_alltz)/2)
result_alltz =result_alltz.reshape(len(result_alltz))
return result_alltz

#x 和 d 样本初始化
train_x =[]
d=[]
# 读取图像，提取每类图像的特征
for ii in xrange(1,picflag+1):
    smp_x=[]
    mytz=np.zeros((3,w_fg*h_fg))
    for jj in xrange(1,4):
        fn='p'+str(ii)+'-'+str(jj)+'.png'
        tmptz=readpic(fn)
        train_x.append(tmptz.tolist())
        d.append(ii)
    x=np.array(train_x)
    y=np.array(d)
    svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly',gamma=50)
    svm.learn(x, y)
    print svm.pred(x)
    fn='ptest3.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s 属于第 %d 类'%(fn,nowi)
    fn='ptest1.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s 属于第 %d 类'%(fn,nowi)
    fn='ptest2.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s 属于第 %d 类'%(fn,nowi)
    fn='ptest21.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s 属于第 %d 类'%(fn,nowi)
    fn='ptest22.png'
    testtz=np.array(readpic(fn))
    nowi=svm.pred(testtz)
    print u'%s 属于第 %d 类'%(fn,nowi)

```

## 10.4 高斯噪声生成

噪声可理解为妨碍人们的感觉器官去正确理解所接收的信源信息的因素,图像中各种会妨碍人们接受其信息的因素都可称为图像噪声,噪声在理论上可以定义为“不可预测,只能用概率统计的方法来认识的随机误差”。图像噪声在数字图像处理技术中的重要性越来越明显。图像噪声按其产生的原因可以分为:

1) 外部噪声。指系统外部的干扰以电磁波或经电源串进系统内部而引起的噪声。如电气设备、天体放电现象等引起的噪声。

2) 内部噪声。一般可分为以下四种:

□ 由光和电的基本性质所引起的噪声。如电流的产生是由电子或空穴粒子的集合,定向运动所形成的。

□ 电器的机械运动产生的噪声。如各种接头因抖动引起电流的变化所产生的噪声;磁头、磁带等抖动或仪器的抖动等。

□ 器材材料本身引起的噪声。如正片和负片的表面颗粒性和磁带磁盘表面的缺陷所产生的噪声。

□ 系统内部设备电路所引起的噪声。如电源引入的交流噪声;偏转系统和箝位电路所引起的噪声等。

既然噪声是不可避免,那么在图像处理的过程中,为检验图像算法的有效性,需要人为地生成一些噪声来模拟现实的环境。

下面以加性零均值高斯噪声为例,通过在灰度图上加上这类高斯噪声进行噪声仿真。具体方法是:在每个点的灰度值上加上一个噪声值,噪声值的产生方式为 Box-Muller 算法生成高斯噪声。Box-Muller 方法是产生随机数的一种方法,算法隐含的原理非常深奥,但是结果却相当简单。它一般是要得到服从正态分布的随机数,基本思想是先得到服从均匀分布的随机数,再将服从均匀分布的随机数转变为服从正态分布。Box-Muller 算法的具体过程如下:

1) 假设图像的灰阶范围是  $[0, G-1]$ 。取  $\sigma > 0$ : 它的值越小时,相应的噪声也越小。

2) 针对每对水平相邻的像素  $(x, y)$ ,  $(x, y+1)$  产生一对位于  $[0, 1]$  的独立随机数  $r$  和  $\Phi$ 。

3) 计算以下值:

$$Z_1 = \sigma \cos(2\pi\Phi) \sqrt{-\ln r}$$

$$Z_2 = \sigma \sin(2\pi\Phi) \sqrt{-\ln r}$$

上式中,  $Z_1$  是  $Z_2$  独立的有 0 均值和  $\sigma$  方差的正态分布。

4) 设  $g$  为输入图像,计算以下值:

$$f'(x, y) = g(x, y) + Z_1$$

$$f'(x, y+1) = g(x, y+1) + Z_2$$

5) 计算  $f(x, y)$  和  $f(x, y+1)$  的值:

$$f(x, y) = \begin{cases} 0 & f'(x, y) < 0 \\ G-1 & f'(x, y) > G-1 \\ f'(x, y) & \text{其他} \end{cases}$$



$$f(x,y+1)=\begin{cases} 0 & f'(x,y+1)<0 \\ G-1 & f'(x,y+1)>G-1 \\ f'(x,y+1) & \text{其他} \end{cases}$$

6) 反复执行第3至第5步,直到像素点处理完毕为止。

下面以 Python 代码来实现对某图像的加高斯噪声的操作,如程序 10-11.py 所示。

```
# -*- coding: utf-8 -*-
# 加性零均值高斯噪声
# code: myhaspl@myhaspl.com
# 10-11.py
import cv2
import numpy as np

fn="test112.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
```

```
param=30
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)
```

```
for x in xrange(0,h):
```

```
    for y in xrange(0,w,2):
```

```
        r1=np.random.random_sample()
```

```
        r2=np.random.random_sample()
```

```
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
```

```
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
```

```
        fxy=int(img[x,y]+z1)
```

```
        fxy1=int(img[x,y+1]+z2)
```

```
        #f(x,y)
```

```
        if fxy<0:
```

```
            fxy_val=0
```

```
        elif fxy>grayscale-1:
```

```
            fxy_val=grayscale-1
```

```
        else:
```

```
            fxy_val=fxy
```

```
        #f(x,y+1)
```

```
        if fxy1<0:
```

```
            fxy1_val=0
```

```
        elif fxy1>grayscale-1:
```

```
            fxy1_val=grayscale-1
```

```
        else:
```

```
            fxy1_val=fxy1
```

```
        newimg[x,y]=fxy_val
```

```
        newimg[x,y+1]=fxy1_val
```

```

cv2.imshow('preview',newimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行程序 10-11.py, 结果如图 10-16 所示。

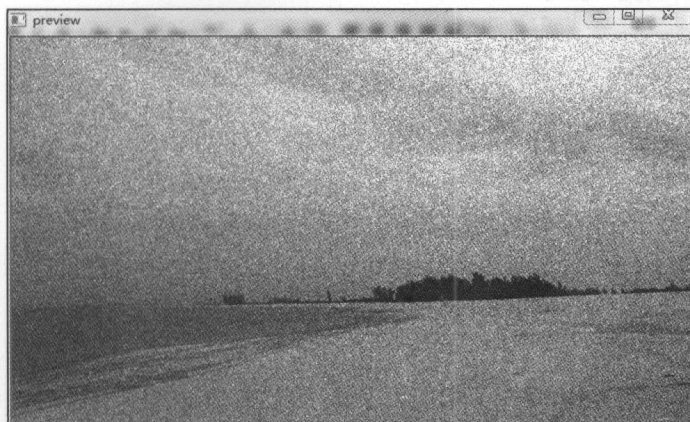


图 10-16 灰度图像高斯噪声

程序 10-12.py 实现了对彩色图像增加高斯噪声, 代码如下所示:

```

# -*- coding: utf-8 -*-
# 加性零均值高斯噪声
#code:myhaspl@myhaspl.com
#10-12.py
import cv2
import numpy as np

```

```

fn="test112.jpg"
myimg=cv2.imread(fn)
img=myimg

param=30
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w,3),np.uint8)

```

```

for x in xrange(0,h):
    for y in xrange(0,w,2):
        r1=np.random.random_sample()
        r2=np.random.random_sample()
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
        fxy_0=int(img[x,y,0]+z1)
        fxy1_0=int(img[x,y+1,0]+z2)
        fxy_1=int(img[x,y,1]+z1)
        fxy1_1=int(img[x,y+1,1]+z2)

```

```

fxy_2=int(img[x,y,2]+z1)
fxy1_2=int(img[x,y+1,2]+z2)
#f(x,y)
if fxy_0<0:
    fxy_val_0=0
elif fxy_0>grayscale-1:
    fxy_val_0=grayscale-1
else:
    fxy_val_0=fxy_0

if fxy_1<0:
    fxy_val_1=0
elif fxy_1>grayscale-1:
    fxy_val_1=grayscale-1
else:
    fxy_val_1=fxy_1

if fxy_2<0:
    fxy_val_2=0
elif fxy_2>grayscale-1:
    fxy_val_2=grayscale-1
else:
    fxy_val_2=fxy_2

#f(x,y+1)
if fxy1_0<0:
    fxy1_val_0=0
elif fxy1_0>grayscale-1:
    fxy1_val_0=grayscale-1
else:
    fxy1_val_0=fxy1_0

if fxy1_1<0:
    fxy1_val_1=0
elif fxy1_1>grayscale-1:
    fxy1_val_1=grayscale-1
else:
    fxy1_val_1=fxy1_1

if fxy1_2<0:
    fxy1_val_2=0
elif fxy1_2>grayscale-1:
    fxy1_val_2=grayscale-1
else:
    fxy1_val_2=fxy1_2

newimg[x,y,0]=fxy_val_0
newimg[x,y,1]=fxy_val_1
newimg[x,y,2]=fxy_val_2
newimg[x,y+1,0]=fxy1_val_0
newimg[x,y+1,1]=fxy1_val_1
newimg[x,y+1,2]=fxy1_val_2

```

```
cv2.imshow('preview',newimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

程序 10-12.py 在彩色图像中人为地增加了若干高斯噪声，运行程序，效果如图 10-17 所示。

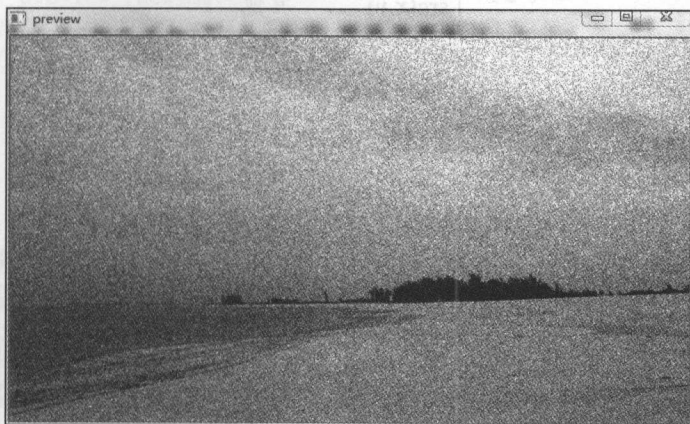


图 10-17 彩色图像加高斯噪声

## 10.5 二值化

图像的二值化，就是将图像上像素点的灰度值设置为 0 或 255，也就是将整个图像呈现出明显的只有黑和白的视觉效果，最常用的方法就是设定一个阈值  $T$ ，用  $T$  将图像的数据分成两部分：大于  $T$  的像素群和小于  $T$  的像素群。

### 10.5.1 threshold

可调用 OpenCV 的 threshold 实现二值化，Python 调用此方法的格式如下：

```
cv2.threshold(src, thresh, maxval, type[, dst]) → retval, dst
```

该方法主要有以下核心参数：

- src: 输入数组（单通道、8 位或 32 位）。
- dst: 二值化输出结果的矩阵。
- thresh: 阈值。
- maxval: 二值化中除 0 以外的其他值。
- type: 二值化类别。主要有以下几种：

- THRESH\_BINARY

$$\text{dst}(x,y) = \begin{cases} \text{maxval} & \text{src}(x,y) > \text{thresh} \\ 0 & \text{其他} \end{cases}$$

- THRESH\_BINARY\_INV

$$\text{dst}(x,y)=\begin{cases} 0 & \text{src}(x,y)>\text{thresh} \\ \text{maxval} & \text{其他} \end{cases}$$

- THRESH\_TRUNC

$$\text{dst}(x,y)=\begin{cases} \text{threshold} & \text{src}(x,y)>\text{thresh} \\ \text{src}(x,y) & \text{其他} \end{cases}$$

- THRESH\_TOZERO

$$\text{dst}(x,y)=\begin{cases} \text{src}(x,y) & \text{src}(x,y)>\text{thresh} \\ 0 & \text{其他} \end{cases}$$

- THRESH\_TOZERO\_INV

$$\text{dst}(x,y)=\begin{cases} 0 & \text{src}(x,y)>\text{thresh} \\ \text{src}(x,y) & \text{其他} \end{cases}$$

编写程序 10-13.py，对某图像进行二值化，效果如图 10-18 所示。

```
#10-13.py
import cv2
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
retval,newimg=cv2.threshold(img,40,255,cv2.THRESH_BINARY)
cv2.imshow('preview',newimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

## 10.5.2 adaptiveThreshold

OpenCV 的 `adaptiveThreshold` 函数可完成自适应二值化，也可以提取边缘。Python 调用此方法的格式如下：

```
cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst]) → dst
```

该函数的核心参数如下：

- ❑ `block_size`：决定局部阈值的 `block` 的大小，`block` 很小时，如 `block_size=3、5、7` 时，表现为边缘提取函数。当把 `block_size` 设为比较大的值时，如 `block_size=21、51` 等，就是二值化。
- ❑ `src`：输入数组（单通道、8 位或 32 位）。
- ❑ `dst`：二值化输出结果的矩阵。
- ❑ `thresholdType`：二值化类型，为 `THRESH_BINARY` 或 `THRESH_BINARY_INV`，见 10.5.1 节对该参数的说明。

编写程序 10-14.py，进行二值化，效果如图 10-19 所示。

```
#10-14.py
import cv2

fn="test3.jpg"
```



```

myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

newimg=cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_
BINARY,5,2)
cv2.imshow('preview',newimg)
cv2.waitKey()
cv2.destroyAllWindows()

```



图 10-18 二值化



图 10-19 adaptiveThreshold 二值化

程序 10-15.py 演示了通过二值化提取边缘，效果如图 10-20 所示。

```

#10-15.py
import cv2

fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

```

### 10.7.2 仿射变换实例

```

newimg=cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_
BINARY,51,2)
cv2.imshow('preview',newimg)
cv2.waitKey()
cv2.destroyAllWindows()

```



图 10-20 二值化提取边缘

## 10.6 插值与缩放

通过 OpenCV 的 `resize` 函数可实现插值与缩放。Python 调用该函数的格式如下：

```
cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]) → dst
```

该函数的 `interpolation` 参数可分别设为 `INTER_NEAREST`（最近邻插值）、`INTER_LINEAR`（双线性插值）、`INTER_AREA`（像素关系重采样）、`INTER_CUBIC`（双立方插值）、`INTER_LANCZOS4`（ $8 \times 8$  像素邻域内 Lanczos 插值）。

程序 10-16.py 演示了插值与缩放的操作，效果如图 10-21 所示。

```
# -*- coding: utf-8 -*-
#10-16.py
```

```
import cv2
```

```
fn="test112.jpg"
img=cv2.imread(fn)
w=img.shape[1]
h=img.shape[0]
```

```
# 放大，双立方插值
newimg1=cv2.resize(img, (w*2,h*2), interpolation=cv2.INTER_CUBIC)
# 放大，最近邻插值
newimg2=cv2.resize(img, (w*2,h*2), interpolation=cv2.INTER_NEAREST)
# 放大，像素关系重采样
newimg3=cv2.resize(img, (w*2,h*2), interpolation=cv2.INTER_AREA)
# 缩小，像素关系重采样
newimg4=cv2.resize(img, (300,200), interpolation=cv2.INTER_AREA)
```

```
cv2.imshow('preview1',newimg1)
cv2.imshow('preview2',newimg2)
cv2.imshow('preview3',newimg3)
cv2.imshow('preview4',newimg4)
cv2.waitKey()
cv2.destroyAllWindows()
```

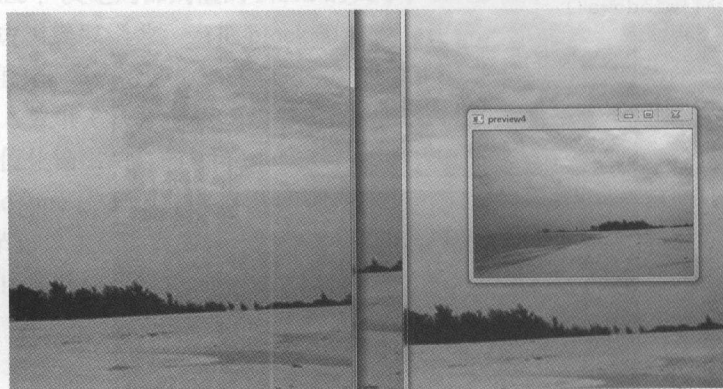


图 10-21 插值与缩放

## 10.7 仿射

### 10.7.1 仿射原理

仿射变换, 又称仿射映射, 是指在几何中, 一个向量空间进行一次线性变换并接上一个平移, 变换为另一个向量空间, 它可对图像进行缩放、旋转、平衡等操作。

一个对向量 $\vec{x}$ 平移 $\vec{b}$ , 与旋转放大缩小 $A$ 的仿射映射为:

$$\vec{y} = A\vec{x} + \vec{b}$$

上式在齐次坐标上, 等价于下面的式子:

$$\begin{bmatrix} \vec{y} \\ 1 \end{bmatrix} = \begin{bmatrix} A & \vec{b} \\ 0, \dots, 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}$$

为了表示仿射变换, 需要使用齐次坐标, 即用三维向量 $(x, y, 1)$ 表示二维向量, 对于高维来说也是如此。按照这种方法, 就可以用矩阵乘法表示变换。 $x'=x+t_x$ ;  $y'=y+t_y$  变为

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

在矩阵中增加一列与一行, 除右下角的元素为 1 外其他部分均填充为 0, 通过这种方法, 所有的线性变换都可以转换为仿射变换。例如, 上面的旋转矩阵可变为

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

通过这种方法, 使用与前面一样的矩阵乘积可以将各种变换无缝地集成到一起。

### 10.7.2 仿射变换实例

#### 1. warpAffine

OpenCV 的 warpAffine 函数可实现仿射变换, Python 调用此方法的格式如下:

```
cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst
```

该函数中的参数 M 表示变换矩阵。函数使用以下变换公式:

$$\text{dst}(x, y) = \text{src}(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

#### 2. getRotationMatrix2D

OpenCV 的 getRotationMatrix2D 函数计算二维旋转变换矩阵, Python 调用此函数的格式如下:

```
cv2.getRotationMatrix2D(center, angle, scale) → retval
```

$$\begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot \text{center} \cdot x - \beta \cdot \text{center} \cdot y \\ -\beta & \alpha & \beta \cdot \text{center} \cdot x + (1-\alpha) \cdot \text{center} \cdot y \end{bmatrix}$$

其中,  $\alpha$  与  $\beta$  的计算公式如下:

$$\alpha = \text{scale} \cdot \cos \text{angle}$$

$$\beta = \text{scale} \cdot \sin \text{angle}$$

程序 10-17.py 演示了仿射变换完成缩小并旋转的操作, 效果如图 10-22 所示。

```
# -*- coding: utf-8 -*-
#10-17.py
import cv2

fn="test3.jpg"
img=cv2.imread(fn)
w=img.shape[1]
h=img.shape[0]
# 得到仿射变换矩阵, 完成旋转
# 中心
mycenter=(h/2,w/2)
# 旋转角度
myangle=90
# 缩放尺度
myscale=0.5
# 仿射变换完成缩小并旋转
transform_matrix=cv2.getRotationMatrix2D(mycenter,myangle,myscale)

newimg=cv2.warpAffine(img,transform_matrix,(w,h))
cv2.imshow('preview',newimg)

cv2.waitKey()
cv2.destroyAllWindows()
```

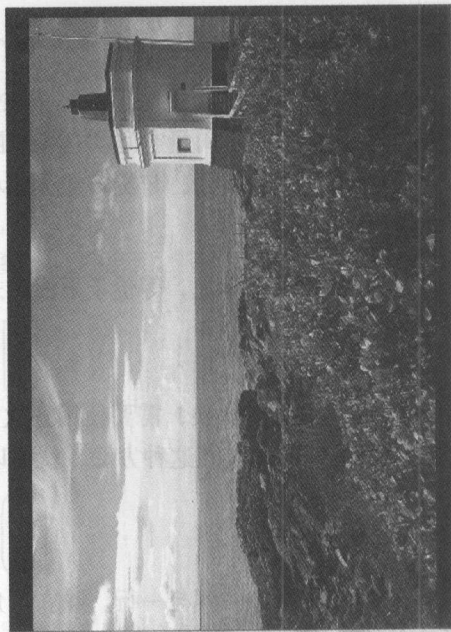


图 10-22 仿射变换

## 10.8 透视投影与透视变换

### 10.8.1 透视投影原理

三维计算机图形学中另外一种重要的变换是透视投影。与平行投影沿着平行线将物体投影到图像平面上不同, 透视投影是指从投影中心这一点发出的直线将物体投影到图像平面上。这就意味着距离投影中心越远的投影越小, 距离越近的投影越大。

最简单的透视投影是将投影中心作为坐标原点,  $z=1$  作为图像平面, 这样投影变换为  $x' = \frac{x}{z}$ ;  $y' = \frac{y}{z}$ , 用齐次坐标表示如下:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

上述乘法的计算结果是  $(x_c, y_c, z_c, w_c) = (x, y, z, z)$ 。乘法计算完毕之后, 通常齐次元素  $w_c$  并



不为 1, 所以为了映射回真实平面需要进行齐次除法, 即每个元素都除以  $w_c$ :

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \end{pmatrix}$$

更加复杂的透视投影可以与旋转、缩放、平移、切变等组合在一起对图像进行变换。

## 10.8.2 透视投影实例

### 1. WarpPerspective

OpenCV 提供了 WarpPerspective 函数, 可对图像进行透视变换。Python 调用此函数的格式如下:

```
cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]]) → dst
```

该函数的主要参数如下:

□ map\_matrix:  $3 \times 3$  变换矩阵。

□ flags: 插值方法和以下开关选项的组合, 有以下两种:

- CV\_WARP\_FILL\_OUTLIERS: 填充所有缩小图像的像素。如果部分像素落在输入图像的边界外, 那么它们的值设定为 fillval。
- CV\_WARP\_INVERSE\_MAP: 指定 matrix 是输出图像到输入图像的反变换, 因此可以直接用来做像素插值。否则, 函数从 map\_matrix 得到反变换。

□ fillval: 用来填充边界外面的值。

该函数对源图像进行转换的计算公式如下:

$$\text{dst}(x,y)=\text{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

### 2. GetPerspectiveTransform

OpenCV 提供了 GetPerspectiveTransform 函数, 以四边形的 4 个点计算透射变换。Python 调用该函数的格式如下:

```
cv2.getPerspectiveTransform(src, dst) → retval
```

该函数的主要参数如下:

□ src: 输入 IDR 的四边形顶点坐标。

□ dst: 输出的相应的四边形顶点坐标。

该函数对  $3 \times 3$  的透射变换矩阵的计算公式如下:

$$\begin{bmatrix} tx'_i \\ ty'_i \\ t_i \end{bmatrix} = \text{map\_matrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

其中,  $\text{dst}(i)=(x'_i, y'_i)$ ,  $\text{src}(i)=(x_i, y_i)$ ,  $i=0,1,2,3$ 。



### 3. 透射变换矩阵计算实例

程序 10-18.py 演示了如何计算透射变换矩阵。

```
# -*- coding: utf-8 -*-
#10-18.py
import cv2
import numpy as np
fn="test112.jpg"
img=cv2.imread(fn)
w=img.shape[1]
h=img.shape[0]
# 得到透射变换矩阵
src=np.array([[0,0],[w-1,0],[w-1,h-1],[0,h-1]], dtype = np.float32)
dst=np.array([[w*0.08,h*0.01],[w*0.8,h*0.25],[w*0.8,h*0.9],[w*0.05,h*0.8]], dtype = np.float32)

transform_matrix=cv2.getPerspectiveTransform(src,dst)
# 输出透射变换矩阵
print transform_matrix

cv2.waitKey()
cv2.destroyAllWindows()
```

运行程序 10-18.py，输出透射变换矩阵，结果如下所示：

```
[[ 8.98634886e-01 -5.70473560e-02 5.12000008e+01]
 [ 1.66413868e-01 7.60111420e-01 3.59999990e+00]
 [ 3.46695559e-04 -1.11420617e-04 1.00000000e+00]]
```

### 4. 透射变换实例

程序 10-19.py 演示了透射变换，效果如图 10-23 所示。

```
# -*- coding: utf-8 -*-
#10-19.py
import cv2
import numpy as np
fn="test112.jpg"
img=cv2.imread(fn)
w=img.shape[1]
h=img.shape[0]
# 得到透射变换矩阵
src=np.array([[0,0],[w-1,0],[w-1,h-1],[0,h-1]], dtype = np.float32)
dst=np.array([[w*0.08,h*0.01],[w*0.8,h*0.25],[w*0.8,h*0.9],[w*0.05,h*0.8]], dtype = np.float32)

transform_matrix=cv2.getPerspectiveTransform(src,dst)
print transform_matrix
# 透射变换完成变形
newimg=cv2.warpPerspective(img,transform_matrix,(w,h))
cv2.imshow('preview',newimg)
cv2.waitKey()
cv2.destroyAllWindows()
```



图 10-23 透射变换

## 10.9 灰度变换与图像增强

### 10.9.1 灰度变换概述

在计算机领域中，灰度（Gray Scale）数字图像是每个像素只有一个采样颜色的图像。这类图像通常显示为从最暗黑色到最亮的白色的灰度，尽管理论上这个采样可以是不同深浅的任何颜色，甚至可以是不同亮度上的不同颜色。灰度图像与黑白图像不同，在计算机图像领域中黑白图像只有黑白两种颜色，灰度图像在黑色与白色之间还有很多级的颜色深度。用于显示的灰度图像通常用每个采样像素 8 位的非线性尺度来保存，这样就可以有 256 种灰度（即  $2^8=256$ ）了。这种精度刚刚能够避免可见的条带失真，并且非常易于编程。灰度图像是一种具有从黑到白 256 级灰度色阶或等级的单色图像。该图像中的每个像素均用 8 位数据表示，因此像素点值介于黑白间的 256 种灰度中的一种。该图像只有灰度等级，没有颜色的变化。

灰度变换是基于点操作的增强方法，它将每一个像素的灰度值按照一定的数学变换公式转换为一个新的灰度值，它能增强图像，扩展图像的对比度，使图像变清晰，使其特征更加突出，灰度非线性变换是指将灰度数据按照经验数据或某种算术非线性关系进行变换后再显示。灰度变换是基于点操作的增强方法，它将每一个像素的灰度值按照一定的数学变换公式转换为一个新的灰度值，如增强处理中的对比度增强。

### 10.9.2 对数变换

对数变换对图像的低亮度区有较大的扩展而对高亮度区进行压缩，简言之就是增强了低值灰度的图像细节，灰度非线性变换公式如下：

$$\text{dst} = \text{Clog}(1 + \text{src})$$

程序 10-20.py 演示了对数变换，效果如图 10-24 所示。

```
#10-20.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
jg_img=np.array(40*np.log(img+1),np.uint8)
cv2.imshow('src',img)
cv2.imshow('dst',jg_img)
cv2.waitKey()
cv2.destroyAllWindows()
```



图 10-24 对数变换

观察图 10-24，左边的是经过非线性变换操作的图，右边的是原图，左边的低亮度区更清晰。

### 10.9.3 分段线性变换

分段线性变换将图像的值域分成多个值域并进行不同的线性变换计算，可以压缩某部分灰度区，扩展另一部分灰度区间，下面以两个区间为例，编写程序进行变换，如程序 10-21.py 所示。

```
# -*- coding: utf-8 -*-
# 分段线性变换
#code:myhaspl@myhaspl.com
#10-21.py
import cv2
import numpy as np
fn="test4.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)
# 源
```

```

Ds_min=0
Ds_internal=80# 中间
Ds_max=255
# 目标
Dd_min=0
Dd_internal=160# 中间
Dd_max=255
for m in xrange(h):
    for n in xrange(w):
        if img[m,n]>Ds_min and img[m,n]<=Ds_internal:
            newimg[m,n]=int((Dd_internal-Dd_min)/(Ds_internal-Ds_min)*(img[m,n]-Ds_min)+Dd_min)
        else:
            newimg[m,n]=int((Dd_max-Dd_internal)/(Ds_max-Ds_internal)*(img[m,n]-Ds_internal)+Dd_internal)
    print ".",
cv2.imshow('src',img)
cv2.imshow('dst',newimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

观察图 10-25, 左边是经过分段线性变换的图像, 右边是原图像, 通过压缩高亮度区, 扩展低亮度区, 使图像的对比度变得更强。

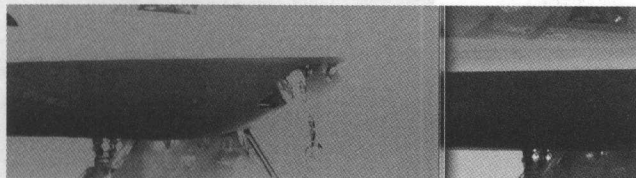


图 10-25 分段线性变换

程序 10-21.py 的代码中, 各变量的含义如下:

- Ds\_min 为源区段的最小值域。
- Ds\_internal 为源区段的中间分界值。
- Ds\_max 为源区段的最大值域。
- Dd\_min 为目标区段的最小值域。
- Dd\_internal 为目标区段的中间分界值。
- Dd\_max 为目标区段的最大值域。

#### 10.9.4 指数变换

指数变换的作用是扩展图像的高灰度级、压缩低灰度级, 可用于亮度过高的图像。指数变换的基本表达式如下:

$$y=b^{c(x-a)}-1$$

其中, 参数  $b$ 、 $c$  控制曲线的变换形状, 参数  $a$  控制曲线的位置。指数变换与对数变换



的关系如图 10-26 所示。

程序 10-22.py 演示了对太阳图像进行指数变换，使低亮度区（温度较低的区域）不再显示，突出亮度区（温度较高的区域），效果如图 10-27 所示。

```
# -*- coding: utf-8 -*-
# 指数非线性变换
#code:myhaspl@myhaspl.com
#10-22.py
import cv2
import numpy as np
fn="test5.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
b=1.2
c=0.2
a=0.2
newimg=np.array(np.power(b,c*(img-a))-1,np.uint8)
cv2.imshow('src',img)
cv2.imshow('dst',newimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

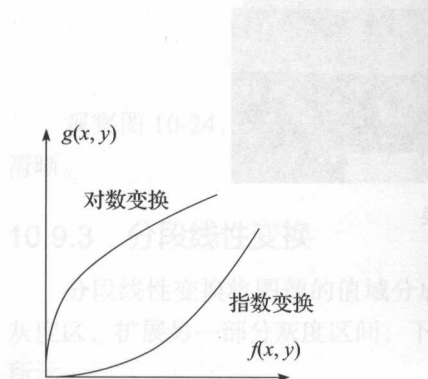


图 10-26 指数变换与对数变换

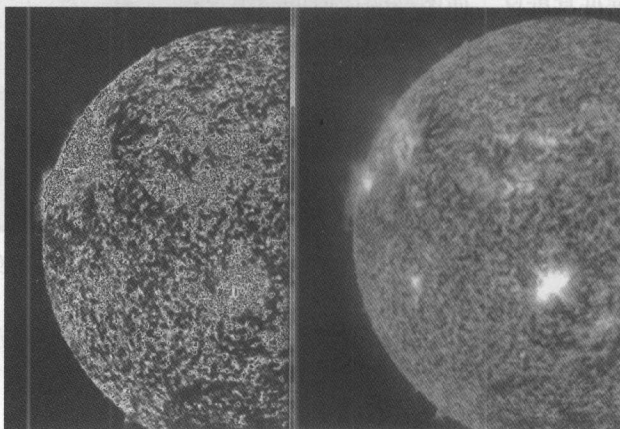


图 10-27 指数变换

观察图 10-27，左边是指数变换后生成的图，右边是原图，左图的高亮度区显示得到突出，达到了预期的效果。

### 10.9.5 直方图均衡化

直方图均衡化通常用来增加许多图像的全局对比度，尤其是当图像的有用数据的对比度相当接近的时候。通过这种方法，亮度可以更好地在直方图上分布。这样就可以用于增强局部的对比度而不影响整体的对比度。在灰度图像上使用直方图均衡化的方法如下：

设有一灰度图像，让  $n_i$  表示灰度  $i$  出现的次数，这样图像中灰度为  $i$  的像素出现的概率如下：



$$p_x(i) = \frac{n_i}{n}, i \in 0, \dots, L-1$$

其中,  $L$  是图像中所有的灰度数,  $n$  是图像中所有的像素数,  $p$  实际上是图像的直方图, 归一化到 0..1。把  $c$  作为对应于  $p$  的累计概率函数, 定义如下:

$$c(i) = \sum_{j=0}^i p_x(j)$$

其中,  $c$  是图像的累计归一化直方图。

然后, 创建一个形式为  $y=T(x)$  的变化, 对于原始图像中的每个值它都产生一个  $y$ , 这样  $y$  的累计概率函数就可以在所有值范围内进行线性化, 转换公式定义如下:

$$y_i = T(x_i) = c(i)$$

上式中,  $T$  将不同的等级映射到 0..1 域, 为了将这些值映射回它们最初的域, 需要在结果上应用下面的简单变换:

$$y'_i = y_i \cdot (\max - \min) + \min$$

此外, 可将上述方法分别用于图像 RGB 颜色值的红色、绿色和蓝色分量, 实现对彩色图像的直方图均衡化。

Python 可调用 OpenCV 的 `equalizeHist` 函数实现直方图均衡化, 调用格式如下:

```
cv2.equalizeHist(src[, dst]) → dst
```

程序 10-23.py 演示了直方图均衡化, 效果如图 10-28 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
#10-23.py
import cv2
fn="test1.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
newimg=cv2.equalizeHist(img)
cv2.imshow('src',img)
cv2.imshow('dst',newimg)
cv2.waitKey()
cv2.destroyAllWindows()
```



图 10-28 直方图均衡化

观察图 10-28，右边是原图，左边是经过增强的图。

再看程序 10-24.py，该程序实现了直方图均衡化的具体算法，效果如图 10-29 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 直方图均衡化
#10-24.py
import cv2
import numpy as np
fn="test5.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
h=img.shape[0]
w=img.shape[1]
newimg=np.zeros((h,w),np.uint8)
scount=0.0
# 原始图像灰度级
scol={}
# 目标图像灰度级
dcol={}
# 原始图像频度
Ps={}
# 累计概率
Cs={}

# 统计原始图像灰度级
for m in xrange(h):
    for n in xrange(w):
        scol[img[m,n]]=scol.setdefault(img[m,n],0)+1
        scount+=1

# 计算原始图像频度
for key in scol:
    Ps[key]=scol[key]/scount

# 计算图像灰度的离散随机变量累计概率
keys=Ps.keys()
keys.sort()
for skey in scol:
    Cs.setdefault(skey,0)
    for key in keys:
        if key>skey:
            break
        Cs[skey]+=Ps[key]

# 建立输入与输出之间的映射
d_max=np.max(keys)
d_min=np.min(keys)
for skey in keys:
    dcol[skey]=int((d_max-d_min)*Cs[skey]+d_min)

for m in xrange(h):
    for n in xrange(w):
```

```

newimg[m,n]=dcol[img[m,n]]

cv2.imshow('src',img)
cv2.imshow('dst',newimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

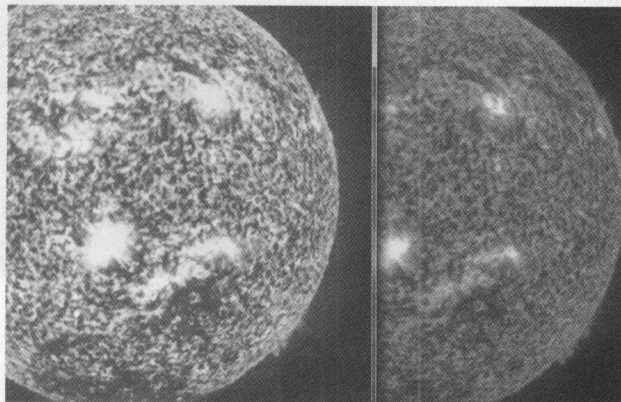


图 10-29 直方图均衡化

观察图 10-29, 左图为均衡化后的图, 右边为原图, 左图的全局对比度增强了。

## 10.10 图像滤波与除噪

### 10.10.1 均一化块滤波

$H(x,y)$  根据作用域 ( $3 \times 3$ 、 $5 \times 5$ 、 $7 \times 7$  等) 的不同而不同。假设为  $3 \times 3$ , 则有 9 个像素参加运算。可以有以下几种:

$$\frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \frac{1}{10} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix} \quad \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

OpenCV 提供的 blur 函数可进行归一化块滤波操作, Python 调用该函数的格式如下:

```
cv2.blur(src, ksize[, dst[, anchor[, borderType]]) → dst
```

此外, 该函数使用了如下脉冲响应函数 (也称为核函数):

$$K = \frac{1}{\text{ksize.width} \times \text{ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

#### 1. 高斯噪声滤波

程序 10-25.py 演示了归一化块滤波对高斯噪声的处理, 效果如图 10-30 所示。

```

# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 归一化块滤波
#10-25.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 加上高斯噪声
param=20
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)

for x in xrange(0,h):
    for y in xrange(0,w,2):
        r1=np.random.random_sample()
        r2=np.random.random_sample()
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))

        fxy=int(img[x,y]+z1)
        fxy1=int(img[x,y+1]+z2)
        #f(x,y)
        if fxy<0:
            fxy_val=0
        elif fxy>grayscale-1:
            fxy_val=grayscale-1
        else:
            fxy_val=fxy
        #f(x,y+1)
        if fxy1<0:
            fxy1_val=0
        elif fxy1>grayscale-1:
            fxy1_val=grayscale-1
        else:
            fxy1_val=fxy1
        newimg[x,y]=fxy_val
        newimg[x,y+1]=fxy1_val

# 滤波去噪
lbimg=cv2.blur(newimg,(3,3))
cv2.imshow('src',newimg)
cv2.imshow('dst',lbimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

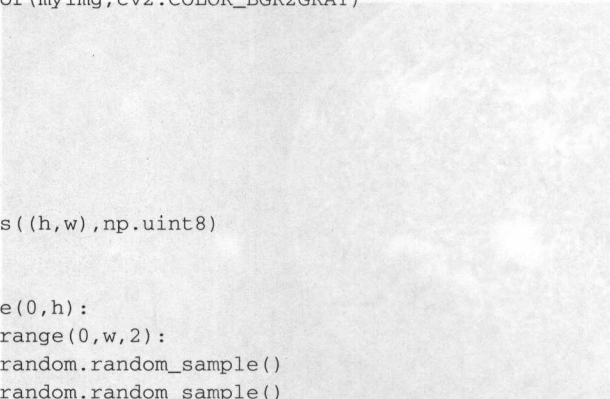




图 10-30 归一化块滤波

观察图 10-30, 左边为经过归一化块滤波后的图, 右边为原图, 滤波效果较好。

当然, 也可以使用第 3 个脉冲响应函数, 如下:

$$\frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

程序 10-26.py 演示了使用第 3 个脉冲响应函数, 应用归一化块滤波对高斯噪声进行处理, 效果如图 10-31 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 归一化块滤波
#10-26.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 加上高斯噪声
param=20
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)

for x in xrange(0,h):
    for y in xrange(0,w,2):
```



```

r1=np.random.random_sample()
r2=np.random.random_sample()
z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
import cv2
fxy=int(img[x,y]+z1)
fxy1=int(img[x,y+1]+z2)
#f(x,y)
if fxy<0:
    fxy_val=0
elif fxy>grayscale-1:
    fxy_val=grayscale-1
else:
    fxy_val=fxy
#f(x,y+1)
if fxy1<0:
    fxy1_val=0
elif fxy1>grayscale-1:
    fxy1_val=grayscale-1
else:
    fxy1_val=fxy1
newimg[x,y]=fxy_val
newimg[x,y+1]=fxy1_val
print "-",

# 滤波去噪
# 图像四个边的像素处理
lbimg=np.zeros((h+2,w+2),np.float32)
tmpimg=np.zeros((h+2,w+2))
myh=h+2
myw=w+2
tmpimg[1:myh-1,1:myw-1]=newimg[0:myh,0:myw]
# 用第 3 个脉冲响应函数
a=1/16.0
kernel=a*np.array([[1,2,1],[2,4,2],[1,2,1]])
for y in xrange(1,myh-1):
    for x in xrange(1,myw-1):
        lbimg[y,x]=np.sum(kernel*tmpimg[y-1:y+2,x-1:x+2])
    print ".",
resultimg=np.array(lbimg[1:myh-1,1:myw-1],np.uint8)
cv2.imshow('src',newimg)
cv2.imshow('dst',resultimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

观察图 10-31，左图为归一化块滤波后的效果，右图为原图，左图去除噪声的效果较好。

## 2. 椒盐噪声滤波

归一化块滤波对椒盐噪声仍有一定的滤波效果，但需要将作用域扩大（比如：设为  $5 \times 5$ ），图像会更模糊，效果较好。

程序 10-27.py 演示了应用归一化块滤波对椒盐噪声进行处理，效果如图 10-32 所示。

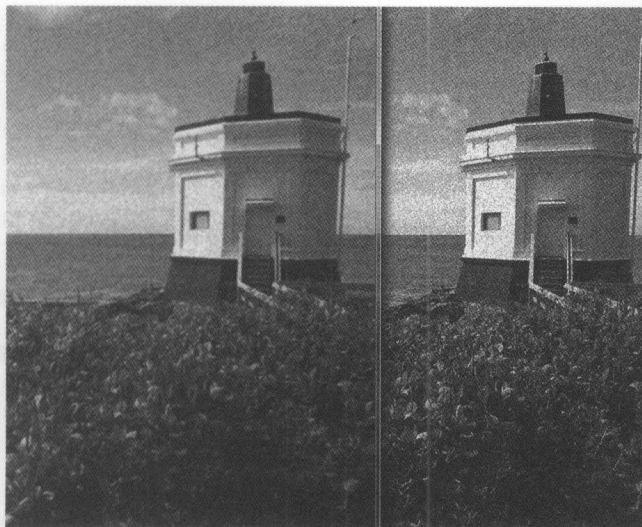


图 10-31 用第 3 个脉冲响应函数归一化块滤波

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 归一化块滤波
#10-27.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
```

```
# 加上椒盐噪声
# 灰阶范围
w=img.shape[1]
h=img.shape[0]
newimg=np.array(img)
# 噪声点数量
noisecount=100000
for k in xrange(0,noisecount):
    xi=int(np.random.uniform(0,newimg.shape[1]))
    xj=int(np.random.uniform(0,newimg.shape[0]))
    newimg[xj,xi]=255
```

```
# 滤波去噪
lbimg=cv2.blur(newimg,(5,5))
cv2.imshow('src',newimg)
cv2.imshow('dst',lbimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

观察图 10-32，左边为经过滤波后的图，右边为原图，去噪效果较好。

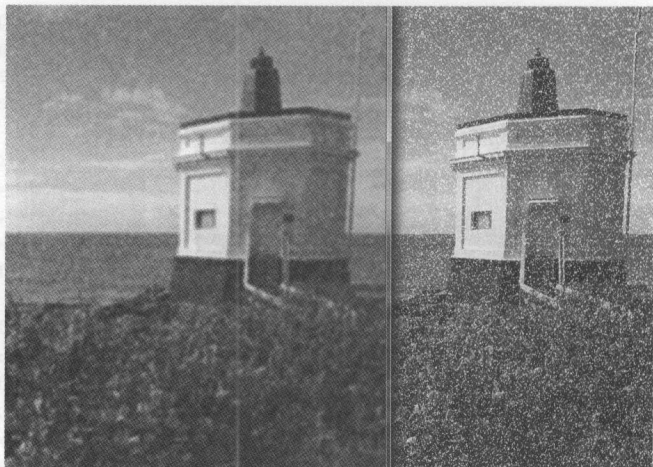


图 10-32 归一化块滤波对椒盐噪声进行处理

### 10.10.2 邻域平均法

邻域平均法可有效消除高斯噪声，其数学公式如下：

$$g(x,y) + \frac{1}{M} \sum_{(k,l) \in S} f(x-k,y-l)$$

$S$  为邻域，不包括  $(x,y)$  本身的像素点，核  $h(x,y)$  可为：

半径为 1：

$$\frac{1}{4} \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

半径为 2：

$$\frac{1}{8} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

程序 10-28.py 演示了邻域平均法对椒盐噪声滤波进行的操作，效果如图 10-33 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 邻域平均法滤波 3*3
#10-28.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 加上椒盐噪声
param=20
# 灰阶范围
```

```

w=img.shape[1]
h=img.shape[0]
newimg=np.array(img)
# 噪声点数量
noisecount=100000
for k in xrange(0,noisecount):
    xi=int(np.random.uniform(0,newimg.shape[1]))
    xj=int(np.random.uniform(0,newimg.shape[0]))
    newimg[xj,xi]=255

# 领域平均法去噪
# 脉冲响应函数,核函数
# 图像四个边的像素处理
lbimg=np.zeros((h+2,w+2),np.float32)
tmpimg=np.zeros((h+2,w+2))
myh=h+2
myw=w+2
tmpimg[1:myh-1,1:myw-1]=newimg[0:myh,0:myw]
# 用领域平均法的(设半径为 2)脉冲响应函数
a=1/8.0
kernel=a*np.array([[1,1,1],[1,0,1],[1,1,1]])
for y in xrange(1,myh-1):
    for x in xrange(1,myw-1):
        lbimg[y,x]=np.sum(kernel*tmpimg[y-1:y+2,x-1:x+2])
    print ".",
resultimg=np.array(lbimg[1:myh-1,1:myw-1],np.uint8)
cv2.imshow('src',newimg)
cv2.imshow('dst',resultimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

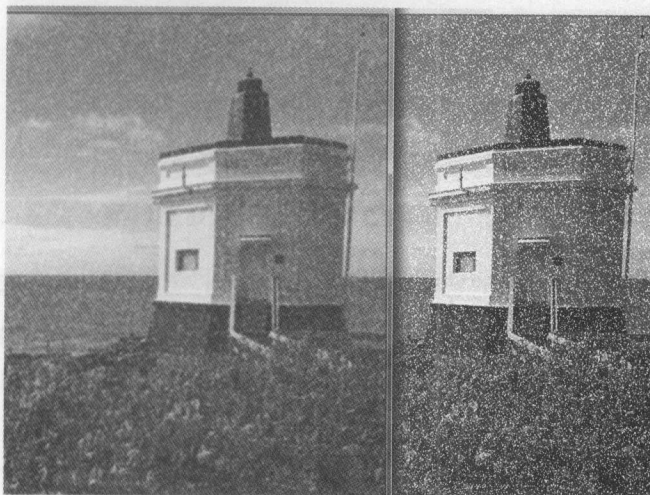


图 10-33 邻域平均法对椒盐噪声滤波

观察图 10-33, 左边为经过滤波后的图, 右边为原图, 滤波效果较好。

程序 10-29.py 演示了邻域平均法对高斯噪声进行滤波的操作，效果如图 10-34 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 邻域平均法滤波
#10-29.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 加上高斯噪声
param=20
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)

for x in xrange(0,h):
    for y in xrange(0,w,2):
        r1=np.random.random_sample()
        r2=np.random.random_sample()
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))

        fxy=int(img[x,y]+z1)
        fxy1=int(img[x,y+1]+z2)
        #f(x,y)
        if fxy<0:
            fxy_val=0
        elif fxy>grayscale-1:
            fxy_val=grayscale-1
        else:
            fxy_val=fxy
        #f(x,y+1)
        if fxy1<0:
            fxy1_val=0
        elif fxy1>grayscale-1:
            fxy1_val=grayscale-1
        else:
            fxy1_val=fxy1
        newimg[x,y]=fxy_val
        newimg[x,y+1]=fxy1_val
    print "-",

# 邻域平均法去噪
# 脉冲响应函数，核函数
# 图像四个边的像素处理
lbimg=np.zeros((h+2,w+2),np.float32)
```



```

tmpimg=np.zeros((h+2,w+2))
myh=h+2
myw=w+2
tmpimg[1:myh-1,1:myw-1]=newimg[0:myh,0:myw]
# 用邻域平均法的(设半径为2)脉冲响应函数
a=1/8.0
kernel=a*np.array([[1,1,1],[1,0,1],[1,1,1]])
for y in xrange(1,myh-1):
    for x in xrange(1,myw-1):
        lbimg[y,x]=np.sum(kernel*tmpimg[y-1:y+2,x-1:x+2])
    print ".",
resultimg=np.array(lbimg[1:myh-1,1:myw-1],np.uint8)
cv2.imshow('src',newimg)
cv2.imshow('dst',resultimg)
cv2.waitKey()
cv2.destroyAllWindows()

```



图 10-34 邻域平均法对高斯噪声滤波

观察图 10-34，左边为经过滤波后的图，右边为原图，滤波效果较好。

### 10.10.3 中值滤波

中值滤波与邻域平均法类似，但计算的是中值，而不是平均值。具体算法是：将图像的每个像素用邻域（以当前像素为中心的长方形区域）像素的中值来代替。

```

# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 中值滤波
#10-30.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)

```

```

img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 加上椒盐噪声
# 灰阶范围
w=img.shape[1]
h=img.shape[0]
newimg=np.array(img)
# 噪声点数量
noisecount=50000
for k in xrange(0,noisecount):
    xi=int(np.random.uniform(0,newimg.shape[1]))
    xj=int(np.random.uniform(0,newimg.shape[0]))
    newimg[xj,xi]=255
# 滤波去噪
# 脉冲响应函数,核函数
# 图像四个边的像素处理
lbimg=np.zeros((h+2,w+2),np.float32)
tmpimg=np.zeros((h+2,w+2))
myh=h+2
myw=w+2
tmpimg[1:myh-1,1:myw-1]=newimg[0:myh,0:myw]
# 用中值法
for y in xrange(1,myh-1):
    for x in xrange(1,myw-1):
        lbimg[y,x]=np.median(tmpimg[y-1:y+2,x-1:x+2])
    print ".",
resultimg=np.array(lbimg[1:myh-1,1:myw-1],np.uint8)
cv2.imshow('src',newimg)
cv2.imshow('dst',resultimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

观察图 10-35 可发现,中值滤波忽略了较高阶灰度和较低阶灰度,直接取中值,可以有效地过滤椒盐噪声。



图 10-35 中值滤波

此外,也可以在 Python 中调用 OpenCV 的 medianBlur 函数实现中值滤波,调用格式如下:

cv2.medianBlur(src, ksize[, dst]) → dst

程序 10-31.py 演示了 medianBlur 函数实现中值滤波的操作过程,效果如图 10-36 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 中值滤波
#10-31.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
```

```
# 加上椒盐噪声
# 灰阶范围
w=img.shape[1]
h=img.shape[0]
newimg=np.array(img)
# 噪声点数量
noisecount=50000
for k in xrange(0,noisecount):
    xi=int(np.random.uniform(0,newimg.shape[1]))
    xj=int(np.random.uniform(0,newimg.shape[0]))
    newimg[xj,xi]=255
```

```
# 滤波去噪
lbimg=cv2.medianBlur(newimg,3)
cv2.imshow('src',newimg)
cv2.imshow('dst',lbimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

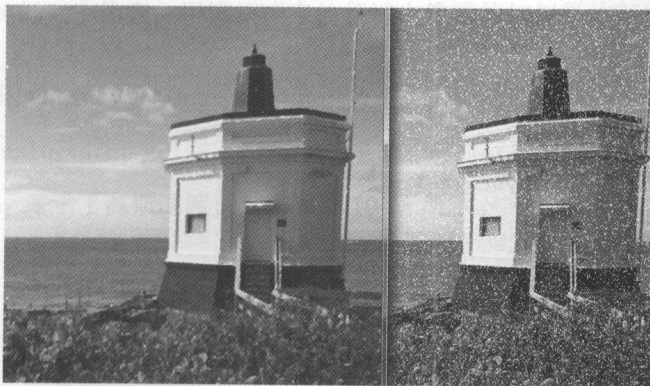


图 10-36 medianBlur 函数实现的中值滤波

观察图 10-36, 左边为经过中值滤波后的图, 右边为原图, 滤波效果很好。

当然，也可以使用中值滤波方法对高斯噪声进行滤波，程序 10-32.py 演示了该操作，效果如图 10-37 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 中值滤波
#10-32.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 灰阶范围
w=img.shape[1]
h=img.shape[0]
newimg=np.array(img)

# 加上高斯噪声
param=20
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)

for x in xrange(0,h):
    for y in xrange(0,w,2):
        r1=np.random.random_sample()
        r2=np.random.random_sample()
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))

        fxy=int(img[x,y]+z1)
        fxy1=int(img[x,y+1]+z2)
        #f(x,y)
        if fxy<0:
            fxy_val=0
        elif fxy>grayscale-1:
            fxy_val=grayscale-1
        else:
            fxy_val=fxy
        #f(x,y+1)
        if fxy1<0:
            fxy1_val=0
        elif fxy1>grayscale-1:
            fxy1_val=grayscale-1
        else:
            fxy1_val=fxy1
        newimg[x,y]=fxy_val
```

```
newimg[x,y+1]=fxy1_val
```

```
# 滤波去噪
lbimg=cv2.medianBlur(newimg,3)
cv2.imshow('src',newimg)
cv2.imshow('dst',lbimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

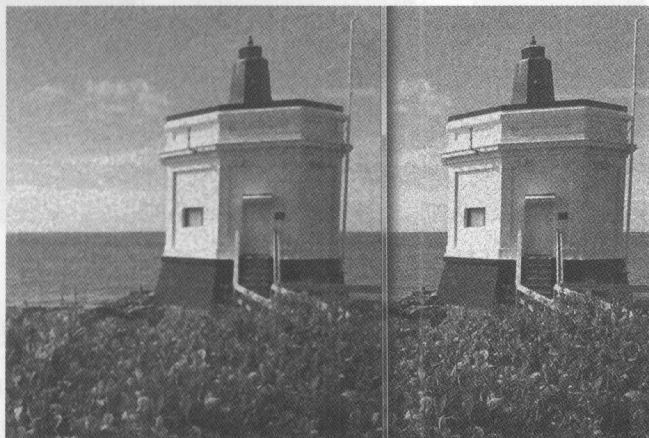


图 10-37 对高斯噪声进行中值滤波

观察图 10-37, 左边为经过中值滤波后的图, 右边为原图, 滤波效果较好。

### 10.10.4 高斯滤波

高斯滤波是对整幅图像进行加权平均的过程, 每一个像素点的值, 都由其本身和邻域内的其他像素值经过加权平均后得到的。高斯滤波的具体操作是: 用一个模板 (或称卷积、掩模) 扫描图像中的每一个像素, 用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

Python 可调用 OpenCV 的 `GaussianBlur` 函数进行高斯滤波, 调用格式如下:

```
cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]]) → dst
```

程序 10-33.py 演示了高斯滤波, 效果如图 10-38 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 高斯滤波
#10-33.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
```



```
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
```

```
# 灰阶范围
```

```
w=img.shape[1]
```

```
h=img.shape[0]
```

```
newimg=np.array(img)
```

```
# 加上高斯噪声
```

```
param=20
```

```
# 灰阶范围
```

```
grayscale=256
```

```
w=img.shape[1]
```

```
h=img.shape[0]
```

```
newimg=np.zeros((h,w),np.uint8)
```

```
for x in xrange(0,h):
```

```
    for y in xrange(0,w,2):
```

```
        r1=np.random.random_sample()
```

```
        r2=np.random.random_sample()
```

```
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
```

```
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
```

```
        fxy=int(img[x,y]+z1)
```

```
        fxy1=int(img[x,y+1]+z2)
```

```
        #f(x,y)
```

```
        if fxy<0:
```

```
            fxy_val=0
```

```
        elif fxy>grayscale-1:
```

```
            fxy_val=grayscale-1
```

```
        else:
```

```
            fxy_val=fxy
```

```
        #f(x,y+1)
```

```
        if fxy1<0:
```

```
            fxy1_val=0
```

```
        elif fxy1>grayscale-1:
```

```
            fxy1_val=grayscale-1
```

```
        else:
```

```
            fxy1_val=fxy1
```

```
        newimg[x,y]=fxy_val
```

```
        newimg[x,y+1]=fxy1_val
```

```
# 滤波去噪
```

```
lbimg=cv2.GaussianBlur(newimg,(3,3),1.8)
```

```
cv2.imshow('src',newimg)
```

```
cv2.imshow('dst',lbimg)
```

```
cv2.waitKey()
```

```
cv2.destroyAllWindows()
```

观察图 10-38，左边为经过滤波后的图，右边为原图，滤波效果较好。

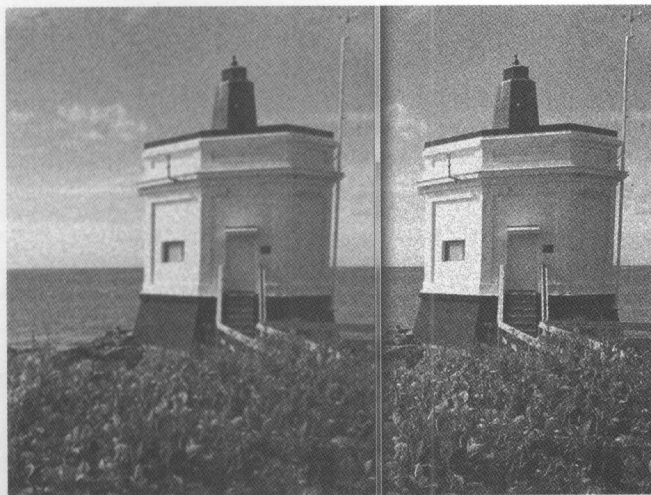


图 10-38 高斯滤波

### 10.10.5 双边滤波

双边滤波 (Bilateral Filter) 是一种非线性的滤波方法, 是结合图像的空间邻近度和像素值相似度的一种折衷处理方法, 同时考虑空域信息和灰度相似性, 达到保边去噪的目的。具有简单、非迭代、局部的特点。

Python 可调用 OpenCV 的 `bilateralFilter` 函数来实现, 调用格式如下:

```
cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace[, dst[, borderType]]) → dst
```

该函数的核心参数如下:

- `d`: 滤波时像素邻域的直径, `d` 为负时由 `sigmaColor` 计算得到; `d>5` 时不能实时处理。
- `sigmaColor`、`sigmaSpace` 表示颜色空间和坐标空间的滤波系数 `sigma`, 可简单的赋值  
为相同的值, 其值小于 10 时几乎没有效果; 其值大于 150 时为油画效果。

程序 10-34.py 演示了双边滤波对椒盐噪声的滤波, 效果如图 10-39 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 双边滤波
#10-34.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
```

```
# 加上椒盐噪声
# 灰阶范围
w=img.shape[1]
h=img.shape[0]
```

```

newimg=np.array(img)
# 噪声点数量
noisecount=50000
for k in xrange(0,noisecount):
    xi=int(np.random.uniform(0,newimg.shape[1]))
    xj=int(np.random.uniform(0,newimg.shape[0]))
    newimg[xj,xi]=255

```

```

# 滤波去噪
lbimg=cv2.bilateralFilter(newimg,5,140,140)
cv2.imshow('src',newimg)
cv2.imshow('dst',lbimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

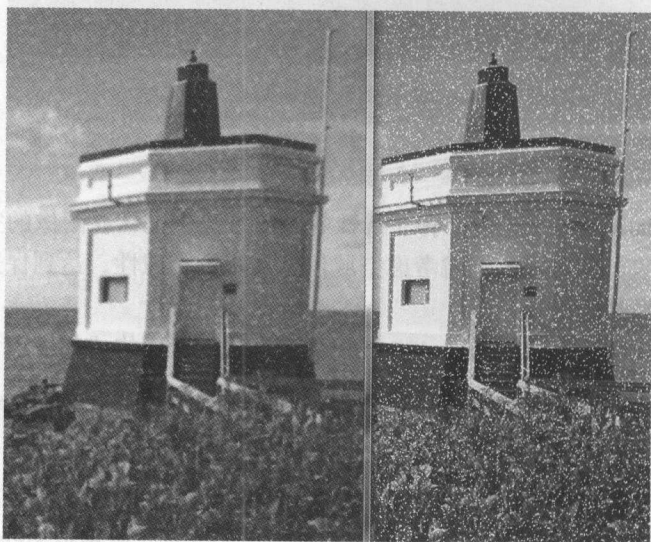


图 10-39 双边滤波对椒盐噪声的滤波

观察图 10-39，左边为经过滤波后的图，右边为原图，滤波效果较好。

程序 10-35.py 演示了双边滤波对高斯噪声的滤波，效果如图 10-40 所示。

```

# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 双边滤波
#11-35.py
import cv2
import numpy as np
fn="test3.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

# 灰阶范围
w=img.shape[1]

```

```

h=img.shape[0]
newimg=np.array(img)

# 加上高斯噪声
param=20
# 灰阶范围
grayscale=256
w=img.shape[1]
h=img.shape[0]
newimg=np.zeros((h,w),np.uint8)

for x in xrange(0,h):
    for y in xrange(0,w,2):
        r1=np.random.random_sample()
        r2=np.random.random_sample()
        z1=param*np.cos(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))
        z2=param*np.sin(2*np.pi*r2)*np.sqrt((-2)*np.log(r1))

        fxy=int(img[x,y]+z1)
        fxy1=int(img[x,y+1]+z2)
        #f(x,y)
        if fxy<0:
            fxy_val=0
        elif fxy>grayscale-1:
            fxy_val=grayscale-1
        else:
            fxy_val=fxy
        #f(x,y+1)
        if fxy1<0:
            fxy1_val=0
        elif fxy1>grayscale-1:
            fxy1_val=grayscale-1
        else:
            fxy1_val=fxy1
        newimg[x,y]=fxy_val
        newimg[x,y+1]=fxy1_val

# 滤波去噪
lbimg=cv2.bilateralFilter(newimg,3,140,140)
cv2.imshow('src',newimg)
cv2.imshow('dst',lbimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

观察图 10-40，左边为经过滤波后的图，右边为原图，滤波效果较好。

### 10.10.6 卷积滤波

卷积滤波的基本思想是：将卷积核矩阵的中心依次放在图像矩阵的每一个像素的位置上，将卷积核的每一个元素分别和图像矩阵对应位置的元素相乘，最终将乘积累加起来，作为卷

积结果。如图 10-41 所示。

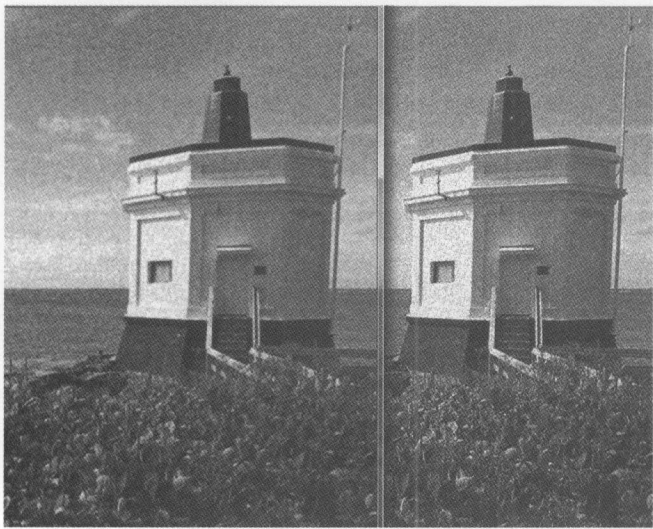


图 10-40 双边滤波对高斯噪声的滤波

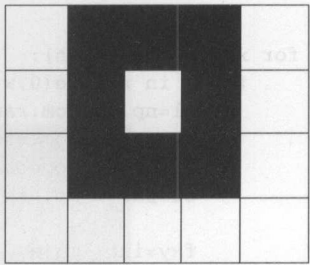


图 10-41 卷积滤波

可在 Python 中使用 OpenCV 的 filter2D 函数进行卷积滤波，调用格式如下：

```
cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]]) → dst
```

程序 10-36.py 演示了卷积滤波对图像的锐化处理，效果如图 10-42 所示。

```
# -*- coding: utf-8 -*-
# 卷积滤波
#code:myhaspl@myhaspl.com
#10-36.py
import cv2
import numpy as np
fn="test112.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
myh=np.array([[0,1,0],[1,-4,1],[0,1,0]])
jgimg=cv2.filter2D(img,-1,myh)
cv2.imshow('src',img)
cv2.imshow('dst',jgimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

观察图 10-42，左边是经过滤波处理的图，右边是原图，锐化效果不错。

程序 10-37.py 演示了拉普拉斯算子进行二维卷积计算，对图像进行锐化处理，效果如图 10-43 所示。

```
# -*- coding: utf-8 -*-
#10-37.py
```



```

import cv2
import numpy as np
from scipy import signal
fn="test6.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)
srcimg=np.array(img,np.double)
myh=np.array([[0,1,0],[1,-4,1],[0,1,0]])
myj=sigal.convolve2d(srcimg,myh,mode="same")
jgimg=img-myj
cv2.imshow('src',img)
cv2.imshow('dst',jgimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

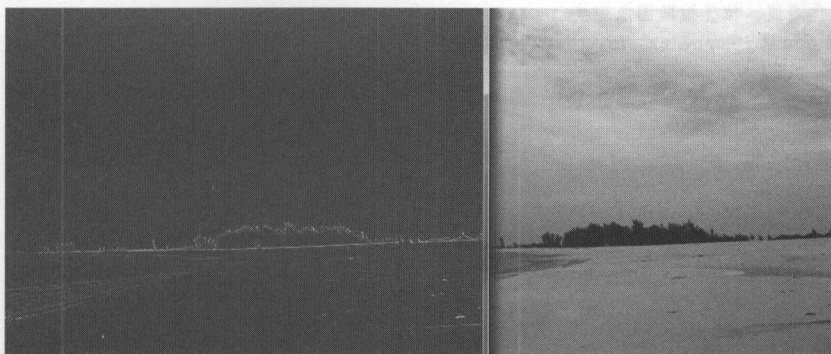


图 10-42 卷积滤波对图像的锐化处理

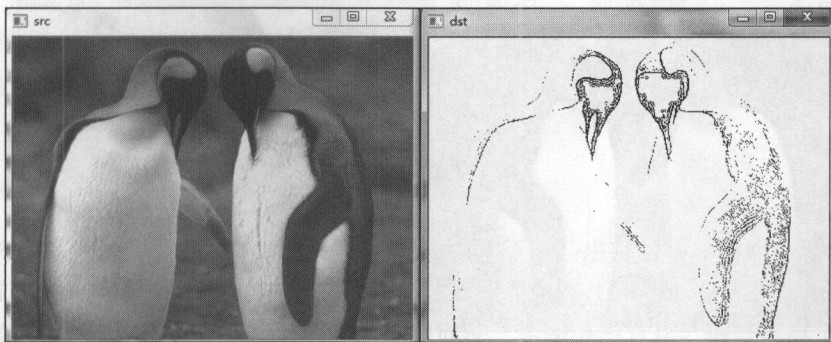


图 10-43 拉普拉斯算子进行二维卷积

观察图 10-43，左边是原图，右边是经过锐化后的图像，锐化效果较好。

### 10.10.7 边缘检测

#### 1. Laplacian

可在 Python 中调用 OpenCV 的 Laplacian 函数进行边缘检测，调用格式如下：

```
cv2.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]])→ dst
```

当该函数的 ksize 参数大于 1 时, Sobel 算子计算公式如下:

$$\text{dst} = \Delta \text{src} = \frac{\partial^2 \text{src}}{\partial x^2} + \frac{\partial^2 \text{src}}{\partial y^2}$$

此外, 当 ksize=1 时, 对图像采用如下内核做卷积:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

程序 10-38.py 演示了拉普拉斯边缘检测, 效果如图 10-44 所示。

```
# -*- coding: utf-8 -*-
# 线性锐化滤波, 拉普拉斯图像变换
# code: myhaspl@myhaspl.com
# 10-38.py
import cv2

fn = "test6.jpg"
myimg = cv2.imread(fn)
img = cv2.cvtColor(myimg, cv2.COLOR_BGR2GRAY)

jgimg = cv2.Laplacian(img, -1)
cv2.imshow('src', img)
cv2.imshow('dst', jgimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

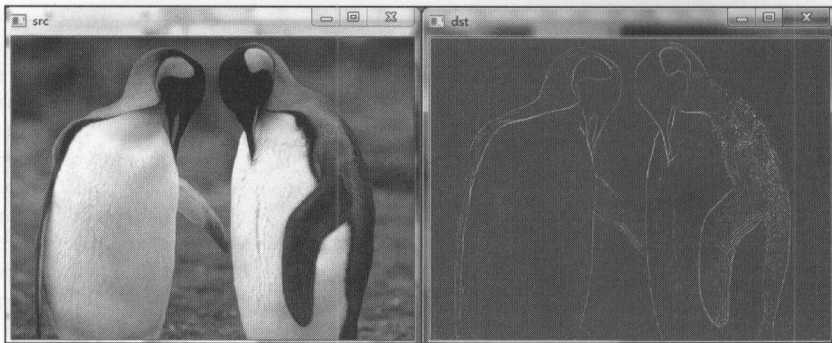


图 10-44 拉普拉斯边缘检测

观察图 10-44, 左边是原图, 右边是经过滤波后的图像, 边缘被成功提取, 将右图进行反转(黑色变白色, 白色变黑色, 用 255 减去图像矩阵)后, 可得到更好的效果。

## 2. sobel 非线性滤波

sobel 非线性滤波采用梯度模的近似方式提取边缘, 锐化图像。

程序 10-39.py 演示了 sobel 非线性滤波, 效果如图 10-45 所示。

```

# -*- coding: utf-8 -*-
# 非线性锐化滤波, sobel 算子变换
# code: myhaspl@myhaspl.com
# 10-39.py
import cv2

fn="test6.jpg"
myimg=cv2.imread(fn)
img=cv2.cvtColor(myimg,cv2.COLOR_BGR2GRAY)

jgimg=cv2.Sobel(img,0,1,1)
cv2.imshow('src',img)
cv2.imshow('dst',jgimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

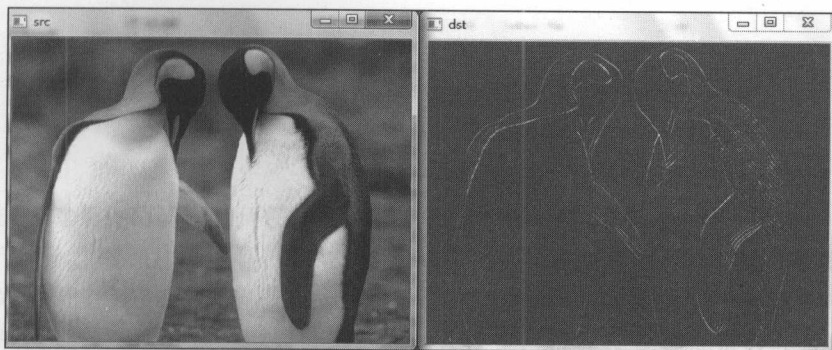


图 10-45 sobel 非线性滤波

观察图 10-45, 左边是原图, 右边是经过滤波后的图像, 边缘被成功提取, 将右图进行反转 (黑色变白色, 白色变黑色, 用 255 减去图像矩阵) 后, 可得到更好的效果。

## 10.11 小结

机器学习算法可对数字图像进行加工和处理, 以便进一步进行特征提取。本章首先介绍了数字图像的基础知识, 数字图像用二维矩阵表示有限像素点, 每个像素点对应于矩阵对应位置的元素; 然后以实例说明了图像边缘、图像匹配、图像分类算法; 最后讲解了噪声生成、图像二值化、插值与缩放、仿射、透视投影与透视变换、灰度变换与图像增强、图像滤波与除噪等算法。此外, 在讲解这些算法的过程中, 注重实践的效果, 每个实例均用 Python 进行实现, 以验证算法的有效性。

## 思考题

- (1) 首先应用欧氏距离算法计算图像边缘, 并对多个不同的图像进行实验, 观察效果;

然后将欧氏距离算法改为像素差分算法（计算像素数值的差分绝对值）进行实验，观察其效果。

(2) 以多个图像为实验对象，对其进行加噪声点、变形、放大、缩小等操作，应用图像匹配技术进行图像匹配算法实验，尝试在图像匹配算法中应用 PCA 降维技术。

(3) 首先任意选取一个彩色图像，分别加上高斯噪声和椒盐噪声，然后应用本章介绍的各种滤波方法，进行滤波除噪，并观察效果。

(4) 首先任意选取一个彩色图像，将其灰度化，然后应用本章介绍的图像增强方法对灰度图像进行处理，最后应用本章介绍的二值化、仿射、透视化变换等方法对彩色图像进行处理。

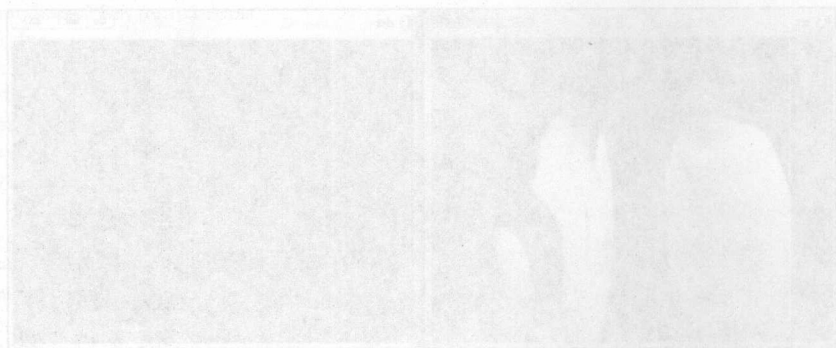


图 10-42 sobel 非线性滤波

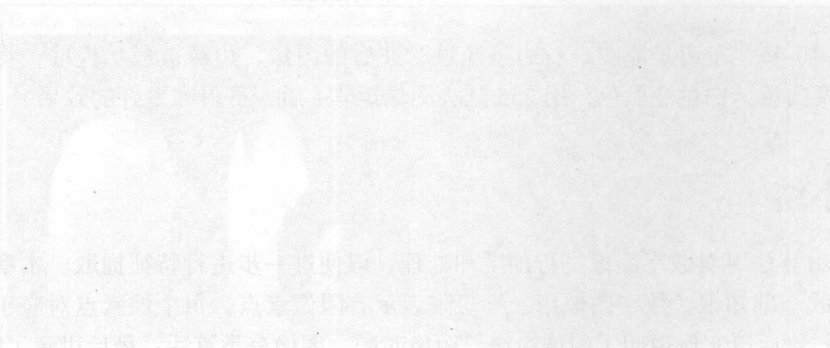


图 10-43 sobel 非线性滤波

## 2. sobel 非线性滤波

图 10-42

图 10-43 显示了 sobel 非线性滤波的结果。可以看到，滤波后的图像与原始图像相比，噪声和失真得到了有效的抑制，图像质量得到了明显的提升。





## 机器视觉案例

计算机视觉是一门研究如何使机器“看懂”图像的科学，即用摄像机和计算机代替人眼对目标进行识别、跟踪和测量等。首先，使用摄像机等设备采集画面，生成数字图像；然后，用计算机对图像进行分析，相当于人的大脑对眼睛采集的信息进行加工，得到所需的信息。

计算机视觉、图像处理、图像分析、机器人视觉、机器视觉等都是彼此紧密关联的学科，这些学科都属于人工智能的范畴，它们的研究目的都是使机器人（计算机）和人一样，能看到图像，理解图像，学习图像中包含的知识。机器学习是人工智能的核心技术，基于机器学习的图像算法应用于图像处理与分析领域，其作用相当于人类大脑加工和处理视觉神经反馈的图像。

### 11.1 人脸辨识

生物特征识别技术，是指通过生物体（一般特指人）本身的生物特征来区分生物体个体。生物特征识别技术所研究的相关特征包括脸、指纹、手掌纹、虹膜、视网膜、声音、体形、个人习惯等。相应的识别技术有人脸识别、指纹识别、掌纹识别、虹膜识别、视网膜识别、语音识别、体形识别、键盘敲击识别、签字识别等。

人脸识别属于生物特征识别技术中的一种，指利用分析比较人脸视觉特征信息进行身份鉴别的计算机技术。

#### 11.1.1 人脸定位

在一张图像或一段视频中定位人脸的技术目前已比较成熟，可调用 OpenCV 提供的接口来完成。相应的 Python 代码如下：



```
def findface(image):
    # 人脸识别, 获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade =
    cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.015,
    2,cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))
    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])
    return result
```

上面算法的原理是：首先将图像转换为灰度图，然后加载 OpenCV 提供的面部特征库，接着调用 HaarDetectObjects 找到人脸的位置，最后将定位的结果赋值给 result 数据并返回。

下面以如图 11-1 所示的《机械公敌》的电影海报为例来应用算法。

实现该算法的完整 Python 代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#11-1.py
# 人脸定位

import cv2
import cv2.cv as cv
print 'loading ...'

# 请在本程序运行前检查 opencv 的目录是否为下面的 OPCV_PATH 值
OPCV_PATH=r"F:/soft/c++/opencv"

def findface(image):
    # 人脸识别, 获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade = cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_
tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.015,
    2,cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))
    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])
    return result

fn='facesb.png'
my_img=cv.LoadImage(fn)

# 获取脸在图像中的坐标
faceresult=findface(my_img)
myimg=cv2.imread(fn)
for ii in xrange(0,len(faceresult)):
```

```

cv2.rectangle(myimg, facerresult [ii][0], facerresult[ii][1], (0,0,250))
cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

程序运行效果如图 11-2 所示。不但所有人类的脸被定位,而且机器人的脸也被成功找到。程序的 OPCV\_PATH 为 OpenCV 源码包中 data 所在的目录,运行 10-10.py 前请自行更改。

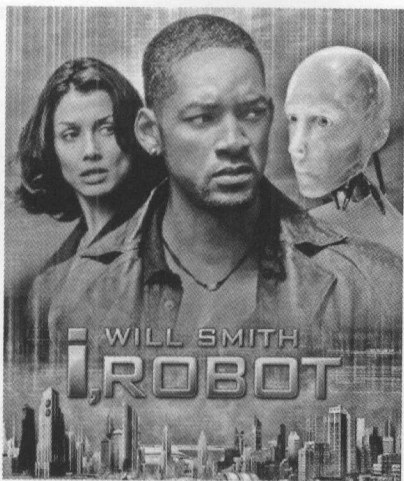


图 11-1 《机械公敌》的电影海报

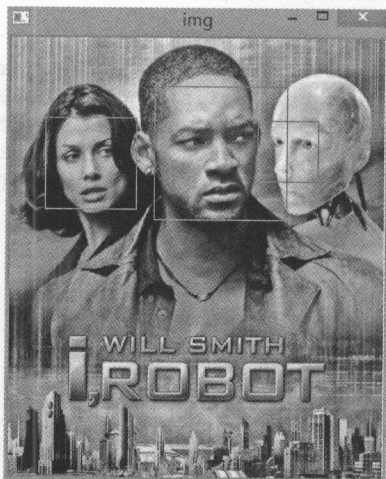


图 11-2 人脸定位

### 11.1.2 人脸辨识

在讲述本节内容之前,先明确一下本节讲述的算法需要完成的任务:通过某人的一张照片,在他与别人的合影中找到他。

#### 1. 算法描述

完成该任务的人脸辨识算法主要过程是:

- 1) 读取两张图像,生成图像矩阵。
- 2) 以两个图像矩阵为基础,调用 OpenCV 的相关函数完成人脸定位。
- 3) 读取两张图像的人脸区域,生成人脸图像矩阵,并将人脸矩阵转换为灰度图。
- 4) 比较分析人脸图像矩阵,找到最相近的人脸。

#### 2. 欧氏距离算法

在进行人脸识别时,可使用标准欧氏距离算法,该算法不仅简单,而且实用。下面以一张 Microsoft 公司员工与比尔·盖茨的合影和在比尔·盖茨办公室中他与某人的合影为例讲解该算法。

算法基本原理是:将标准欧氏距离算法作为比较分析人脸图像矩阵方法。首先,将两个人脸图像调整为指定大小;接着,用所包含像素的三元色数值组成特征组,然后将特征组映

射为高维空间的某个点(在此称之为特征点);最后,计算两个人脸图像的特征点映射到高维空间后的距离,以欧氏距离最小者为最匹配的人脸。具体步骤如下。

1) 调用 OpenCV 相关函数定位人脸。代码如下:

```
cv2.imshow('img', myimg)
cv2.waitKey()
cv2.destroyAllWindows()
```

2) 计算欧氏距离。代码如下:

```
def get_distance(img, finding):
    newsize=(img.shape[1],img.shape[0])
    fimg=cv2.resize(finding,newsize)
    my_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    return get_EuclideanDistance(my_img,my_fimg)
```

3) 找到最匹配的人脸。代码如下:

```
myimg=cv2.imread(fn)
myimgt=cv2.imread(fnt)

#IT 精英比尔·盖茨
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
```

完整的代码如下:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#11-2.py
# 标准欧氏距离实现的人脸识别
```

```
import cv2
import numpy as np
import cv2.cv as cv
print 'loading ...'
# 请在本程序运行前检查 opencv 的目录是否为下面的 OPCV_PATH 值
OPCV_PATH=r"F:/soft/c++/opencv"
def get_EuclideanDistance(x,y):
```

```

myx=np.array(x)
myy=np.array(y)
return np.sqrt(np.sum((myx-myy)*(myx-myy))*np.var(myx-myy))

def get_distance(img, finding):
    newsize=(img.shape[1],img.shape[0])
    fimg=cv2.resize(finding,newsize)
    my_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    return get_EuclideanDistance(my_img,my_fimg)

def findface(image):
    # 人脸识别, 获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade = cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_
tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.1,
2,cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))

    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])
    return result

fn='billall1.png'
fnt= 'billtest.png'
my_img=cv.LoadImage(fn)
face_test=cv.LoadImage(fnt)

# 获取人脸在图像中的坐标
faceresult=findface(my_img)
facet_result=findface(face_test)

myimg=cv2.imread(fn)
myimgt=cv2.imread(fnt)

#IT 精英比尔·盖茨
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1]
[0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_
result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1]
[0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_
result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

cv2.namedWindow('img')
cv2.imshow('img', myimg)
cv2.namedWindow('imgt')

```



```
cv2.imshow('imgt', myimgt)
cv2.waitKey()
cv2.destroyAllWindows()
```

运行算法程序后，成功地在员工合影中找到比尔·盖茨。运行本书资源包中的上述代码，可验证人脸识别效果。

### 3. 改进的欧氏距离算法

前面描述的图像特征码提取算法仅仅是基于像素点的三元色数值的，有时候，图像少量的像素点差异可能干扰识别结果。人脸是一个整体，可能因为人脸的某个部位（如文身、饰品等）、人脸的不同动作和不同角度（如正面人脸、斜侧面人脸以及抬头或低头等）造成少数像素点之间的差异过大，使欧式距离失真（被放大或缩小）。如果从以下两个方面改进上述人脸识别算法，可以使其识别效果更好。

- 将人脸图像大小设置为适当的数值（通常来说比较小），这样更能突出人脸的特征，而略去很多干扰项。此外，提取原始特征组后，使用 PCA 降维技术对原始特征组进行进一步加工，生成最终的特征组，从而更好地表征人脸。
- 在标准欧氏距离的基础上乘以权重。有两种计算方式：第一，每区域像素设置不同的权重，因为人脸的不同区域表征人脸的能力不同；第二，设置整体权重，使人脸图像矩阵的差分值均匀化。

本节使用第二种方式，加工后的欧氏距离不但能更好地表征人脸整体差异，而且不会因为人脸中某些部分过大或过小的差异影响整体识别效果。

在讲解上述算法之前，先讲解一下统计学的变异系数。标准差和方差均可反映数据离散程度，但反映的是离散绝对值。在衡量数据分布离散程度时，不仅要考虑变量值离散程度，还要考虑变量值平均水平。变异系数同时考虑了这两个因素。

变异系数，又称离散系数，是概率分布离散程度的一个归一化量度。它定义为标准差  $\sigma$  与平均值  $\mu$  之比：

$$C_v = \frac{\sigma}{\mu}$$

注意，变异系数只在平均值不为零时有定义，人脸差分矩阵正好满足这个条件。

变异系数可以消除因为平均数不同在变异程度比较中产生的干扰。变异系数越小，数据离平均值的偏离程度越小；反之，变异系数越大，数据离平均值的偏离程度越大。

这里首先将变异系数改进，将标准差用方差代替；然后将改进的变异系数的倒数作为计算欧氏距离的调节系数（这样做的效果是：将偏离程度较大的数据赋予较小的权重，将偏离程度较小的数据赋予较大的权重中）；最后将标准欧氏距离乘以调节权重，从而实现差异平均化，让改进后的欧氏矩阵更好地表征人脸整体差异。通过对多个样本的实验来看，这种方式的效果比较理想。

下面以找到威尔·史密斯为例来讲解改进的欧氏距离算法。在《机械公敌》中，威尔·史密斯饰演警探戴尔史普纳，布丽姬穆娜饰演苏珊卡尔文博士。以威尔·史密斯出演的



《我是传奇》的电影海报(如图 11-3 所示)中的图像为蓝本,在他与布丽姬穆娜合影的《机械公敌》海报(如图 11-4 和图 11-2 所示)中找到他。

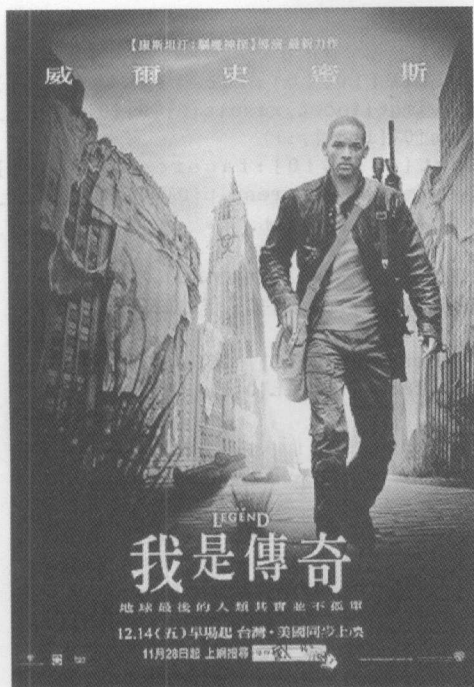


图 11-3 《我是传奇》海报



图 11-4 《机械公敌》海报

经过实验,发现标准的欧氏距算法无法完成这个任务,需要使用刚刚描述的改进后的欧氏距离算法。

1) 在 PCA 降维后,计算基于整体权重的欧氏距离。代码如下:

```
def get_distance(img, findimg):
    newsize=(21,21)
    fimg=cv2.resize(findimg,newsize)
    img=cv2.resize(img,newsize)
    my_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    pcaimg = mlpy.PCA()
    pcaimg.learn(my_img)
    pca_img = pcaimg.transform(my_img, k=1)
    pca_img=pcaimg.transform_inv(pca_img)
    pcafimg = mlpy.PCA()
    pcafimg.learn(my_fimg)
    pca_fimg = pcaimg.transform(my_fimg, k=1)
    pca_fimg= pcafimg.transform_inv(pca_fimg)
    return get_EuclideanDistance(pca_img,pca_fimg)
```

2) 根据欧氏距离决定哪个人脸更匹配。代码如下:

```

my_img=cv.LoadImage(fn2)
myimg=cv2.imread(fn2)
# 获取脸在图像中的坐标
faceresult=findface(my_img)

# 找到威尔·史密斯
isface1=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1][0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

```

完整的代码如下：

```

#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#11-3.py
#pcb+ 改进变异系数人脸识别

import cv2
import numpy as np
import cv2.cv as cv
import mlpy
print 'loading ...'
# 请在本程序运行前检查 opencv 的目录是否为下面的 OPCV_PATH 值
OPCV_PATH=r"F:/soft/c++/opencv"
def get_EuclideanDistance(x,y):
    myx=np.array(x)
    myy=np.array(y)
    return np.sqrt(np.sum((myx-myy)*(myx-myy)))/(np.var(myx-myy)/abs(np.mean(myx-myy)))

def get_distance(img,finding):
    newsiz=(21,21)
    fimg=cv2.resize(finding,newsiz)
    img=cv2.resize(img,newsiz)
    my_img=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    my_fimg=cv2.cvtColor(fimg,cv2.COLOR_BGR2GRAY)
    pcaimg = mlpy.PCA()
    pcaimg.learn(my_img)
    pca_img = pcaimg.transform(my_img, k=1)
    pca_img=pcaimg.transform_inv(pca_img)

    pcafimg = mlpy.PCA()
    pcafimg.learn(my_fimg)

```

```

pca_fimg = pcaimg.transform(my_fimg, k=1)
pca_fimg= pcaimg.transform_inv(pca_fimg)
return get_EuclideanDistance(pca_img,pca_fimg)

def findface(image):
    # 人脸识别, 获取脸在图像中的坐标
    grayscale = cv.CreateImage((image.width, image.height), 8, 1)
    cv.CvtColor(image, grayscale, cv.CV_BGR2GRAY)
    cascade = cv.Load(OPCV_PATH+"/data/haarcascades/haarcascade_frontalface_alt_
tree.xml")
    rect = cv.HaarDetectObjects(grayscale, cascade, cv.CreateMemStorage(), 1.1,
2,cv.CV_HAAR_DO_CANNY_PRUNING, (10,10))
    result = []
    for r in rect:
        result.append([(r[0][0], r[0][1]), (r[0][0]+r[0][2], r[0][1]+r[0][3])])

    return result
fn1='jjgdall.png'
fn2='facesb.png'
fnt='jjgdttest.png'
face_test=cv.LoadImage(fnt)
# 获取人脸在图像中的坐标
facet_result=findface(face_test)
myimgt=cv2.imread(fnt)
my_img=cv.LoadImage(fn1)
myimg=cv2.imread(fn1)
# 获取人脸在图像中的坐标
faceresult=findface(my_img)
# 找到威尔·史密斯
isfacel=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1]
[0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_
result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1]
[0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_
result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isfacel<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

cv2.namedWindow('img1')
cv2.imshow('img1', myimg)
my_img=cv.LoadImage(fn2)
myimg=cv2.imread(fn2)
# 获取人脸在图像中的坐标
faceresult=findface(my_img)

# 找到威尔·史密斯
isfacel=get_distance(myimg[faceresult[0][0][0]:faceresult[0][1]
[0],faceresult[0][0][1]:faceresult[0][1][1],:],myimgt[facet_result[0][0][0]:facet_
result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])

```

```
isface2=get_distance(myimg[faceresult[1][0][0]:faceresult[1][1][0],faceresult[1][0][1]:faceresult[1][1][1],:],myimgt[facet_result[0][0][0]:facet_result[0][1][0],facet_result[0][0][1]:facet_result[0][1][1],:])
if isface1<isface2:
    cv2.rectangle(myimg, faceresult[0][0], faceresult[0][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))
else:
    cv2.rectangle(myimg, faceresult[1][0], faceresult[1][1],(255,0,255))
    cv2.rectangle(myimgt, facet_result[0][0], facet_result[0][1],(255,0,255))

cv2.namedWindow('img2')
cv2.imshow('img2', myimg)
cv2.namedWindow('imgt')
cv2.imshow('imgt', myimgt)
cv2.waitKey()
cv2.destroyAllWindows()
```

运行上述程序后，效果良好，在图 11-5 中，以左图中的史密斯的脸为依据，成功地在右边两个图像中找到史密斯（方框中标示了人脸）。

11.2 手写数字识别

手写数字识别就是指用计算机读取含有手写数字的图像，然后将图像转换为用计算机文字表示的对应数字。SVM 在文本分类、手写文字识别、图像分类、生物序列分析等实际应用中表现出了非常好的性能。下面应用 SVM 算法对 1 ~ 9 的手写数字图像进行识别。



图 11-5 人脸匹配

11.2.1 手写数字识别算法

1) 为每个数字各准备 4 个样本图像，如图 11-6 所示。

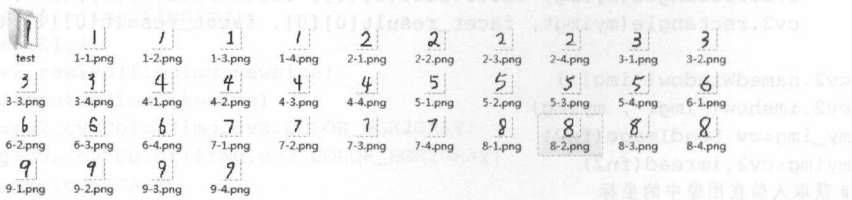


图 11-6 数字样本图像

2) 将图像大小调整为 8×8 的尺寸，读取样本图像。  
虽然较大的图像比较小的图像尺寸更清晰，但并不意味着它能更好地表现图像的主要特



征。不过，由于手写字体不规范，因此较大的尺寸会更清楚地表征字体的不规范细节。当图像尺寸调整得更小时，图像矩阵规模变小，能表现图像细节的能力减弱，大部分字体的不规范细节会消失，但这样一来，数字的主要特征就更加突出了，可尽可能消除手写字体不规范带来的影响。比如，如图 11-7 所示的数字 3，从左边起第一张是  $40 \times 40$  的尺寸，第二张是  $20 \times 20$  的尺寸，而第三张则是  $8 \times 8$  的尺寸。我们在用肉眼观察时，可以看到，随着尺寸越来越小，图像越来越模糊，不规范的细节也慢慢消失。当计算机使用  $8 \times 8$  的尺寸来分析数字 3 的图像时，提取的特征就能更好地表达图像。

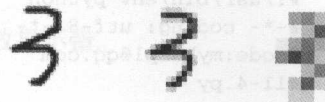


图 11-7 不同尺寸的数字 3 图像

在  $8 \times 8$  (实际是  $8 \times 8 \times 3$ ，因为调用 OpenCV 库读取图像后，会多出一维空间，多出的这维空间分别放置了红、绿、蓝三元色数值) 的图像矩阵中，当色彩为黑色(数字字体的颜色)时，表示该像素的特征码为 1，否则为 0，形成的像素特征码组成了样本图像特征组。

现在将每个数字类别的样本图像特征组作为输入，样本对应的数字值作为输出，用 SVM 进行训练。

3) 读取未知样本图像，将未知图像调整为  $8 \times 8$  的尺寸，提取图像特征码，生成图像特征组。

4) 将未知样本图像特征组送入 SVM 仿真测试，仿真输出值为根据图像识别出的数字。

## 11.2.2 算法的 Python 实现

现在根据上述步骤一步步来实现。首先，提取图像特征，代码如下：

```
def getnumc(fn):
    ''' 返回数字特征 '''
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(8,8))
    alltz=[]

    for now_h in xrange(0,8):
        xtz=[]
        for now_w in xrange(0,8):
            b = img[now_h,now_w,0]
            g = img[now_h,now_w,1]
            r = img[now_h,now_w,2]
            btz=255-b
            gtz=255-g
            rtz=255-r
            if btz>0 or gtz>0 or rtz>0:
                nowtz=1
            else:
                nowtz=0
            xtz.append(nowtz)
        alltz+=xtz
    return alltz
```

然后训练 SVM 并仿真输出。代码如下：



```
x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly',gamma=10)
svm.learn(x, y)
print svm.pred(x)
```

完整的 Python 代码如下:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#11-4.py
```

```
import numpy as np
import mlpy
import cv2
print 'loading ...'
```

```
def getnumc(fn):
    ''' 返回数字特征 '''
    fnimg = cv2.imread(fn)
    img=cv2.resize(fnimg,(8,8))
    alltz=[]
    for now_h in xrange(0,8):
        xtz=[]
        for now_w in xrange(0,8):
            b = img[now_h,now_w,0]
            g = img[now_h,now_w,1]
            r = img[now_h,now_w,2]
            btz=255-b
            gtz=255-g
            rtz=255-r
            if btz>0 or gtz>0 or rtz>0:
                nowtz=1
            else:
                nowtz=0
            xtz.append(nowitz)
        alltz+=xtz
    return alltz
```

# 读取样本数字

```
x=[]
y=[]
for numi in xrange(1,10):
    for numij in xrange(1,5):
        fn='nums/'+str(numi)+'-'+str(numij)+'.png'
        x.append(getnumc(fn))
        y.append(numi)
```

```
x=np.array(x)
y=np.array(y)
svm = mlpy.LibSvm(svm_type='c_svc', kernel_type='poly',gamma=10)
svm.learn(x, y)
```

```
print svm.pred(x)

for iii in xrange (1,10):
    testfn= 'nums/test/'+str(iii)+'-test.png'
    testx=[]
    testx.append(getnumc(testfn))
    print svm.pred(testx)
```

最后,进行仿真测试。

通过对如图 11-8 所示的未知样本的仿真测试得知,运行效果不错。

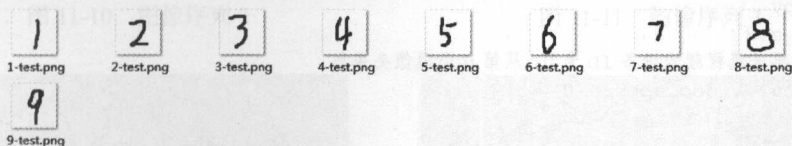


图 11-8 未知数字样本

运行效果如下:

```
loading ...
训练样本测试
[ 1.  1.  1.  1.  2.  2.  2.  2.  3.  3.  3.  3.  4.  4.  4.  4.  5.  5.
  5.  5.  6.  6.  6.  6.  7.  7.  7.  7.  8.  8.  8.  8.  9.  9.  9.  9.]
未知图像测试
nums/test/1-test.png: [ 1.]
nums/test/2-test.png: [ 2.]
nums/test/3-test.png: [ 3.]
nums/test/4-test.png: [ 4.]
nums/test/5-test.png: [ 5.]
nums/test/6-test.png: [ 6.]
nums/test/7-test.png: [ 7.]
nums/test/8-test.png: [ 8.]
nums/test/9-test.png: [ 9.]
```

### 11.3 运动侦测

运动侦测,英文翻译为“Motion Detection Technology”,一般也叫移动检测,是指通过摄像头按照不同的帧率采集得到的图像,电脑按照一定的算法对这些图像进行计算和比较,当画面有变化时,指示系统能自动做出相应的处理。比如:有人走过,镜头被移动,电脑计算比较得出的移动数值,检测是否会超过阈值,如果超过阈值,则进行下一步处理(发出警报等)。

移动侦测技术是运动检测录像技术的基础,最早用于于无人值守监控录像和自动报警,现在已经被广泛应用于网络摄像机、汽车监控锁、数字宝护神、婴儿监视器、自动取样仪、自识别门禁等众多安防仪器和设施上。

### 11.3.1 视频采集

#### 1. 视频采集

程序 11-5.py 演示了视频采集，程序每隔 15 毫秒检测一次是否采集图像。程序的具体实现方式如下：

首先设置捕获设备，然后建立循环，每 15 毫秒更新一次画面显示，并检测是否退出，或采集图像，按空格键则退出，按 c 键则捕获当前视频的图像。

```
# -*- coding: utf-8 -*-
#coding:myhaspl@myhaspl.com
#11-5.py
import cv2
# 设置需要采集视频的设备 ID 为 0，从第 0 个摄像头采集
mycap=cv2.VideoCapture(0)
id=0
while True:
    ret,im=mycap.read()
    cv2.imshow('myvideo',im)
    # 每 15 毫秒采集 1 次
    key=cv2.waitKey(15)
    # 空格键退出
    if key==32:
        break
    #c 键采集
    elif key==ord('c'):
        cv2.imwrite('vd_'+str(id)+'.png',im)
        id+=1
```

程序 11-5.py 运行后，将出现从摄像头实时采集的图像，如图 11-9 所示。



图 11-9 摄像头实时采集的图像 1

每次按 c 键后，采集连接图像，如图 11-10 至图 11-13 所示的图像序列。众



图 11-10 图像序列 2

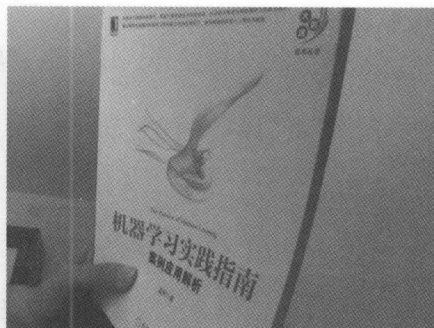


图 11-11 图像序列 3

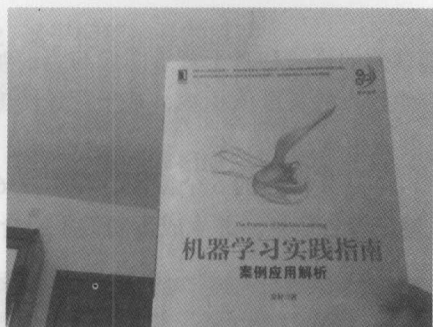


图 11-12 图像序列 4

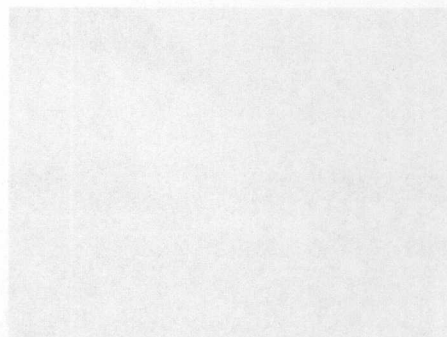


图 11-13 图像序列 5

## 2. 图像序列数组

可按帧生成视频图像序列数组，程序 11-6.py 演示了对视频图像的抓图保存。

```
# -*- coding: utf-8 -*-
#coding:myhaspl@myhaspl.com
# 采集视频生成视频帧数组
#11-6.py
import cv2
# 设置需要采集视频的设备 ID 为 0，从第 0 个摄像头采集
mycap=cv2.VideoCapture(0)
myframes=[]
while True:
    ret,im=mycap.read()
    cv2.imshow('myvideo',im)
    # 采集
    myframes.append(im)
    # 每 15 毫秒采集 1 次
    key=cv2.waitKey(15)
    # 空格键退出
    if key==32:
        break
# 生成视频帧数组
myframes=np.array(myframes)
```





### 3. 读取视频文件

除了从摄像头采集数据外，还可从视频文件获取视频，用如下格式的 Python 代码来实现：

```
mycapture=cv2.VideoCapture(" 文件名 ")
```

## 11.3.2 差分算法

### 1. 差分检测

差分检测根据当前图像与参考图像的差别分析来判断序列图像中是否有运动的物体，在环境亮度变化不大的情况下，如果对应像素灰度值的差异小于某个阈值，则认为画面静止无运动变化，如果图像区域某处的灰度变化大于某个阈值，则认为这是由于图像中运动的物体所引起的，然后求出运动目标在图像中的位置。

根据差分的参考图像不同，可以分为基于相邻帧差的算法和基于背景图像与当前帧差的算法：基于相邻帧差的算法是将前后两帧图像对应像素点的灰度值相减，基于背景图像与当前帧差的算法则是将当前帧和背景帧对应像素的灰度值相减。

下面将以实例来说明该算法。图 11-14 ~ 图 11-19 是某视频的 6 个截图，根据这 6 个运动图像序列，判断画面在何时出现了运动的物体，输出警告信息，并生成运动效果组合图。

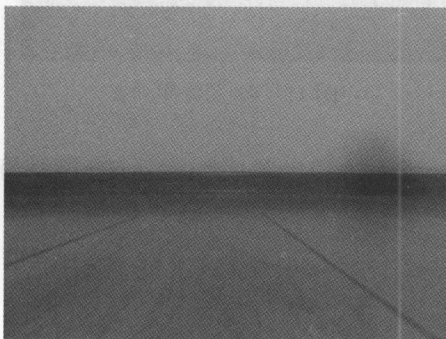


图 11-14 视频的截图 1



图 11-15 视频的截图 2

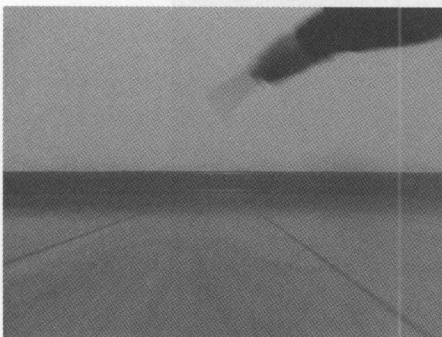


图 11-16 视频的截图 3

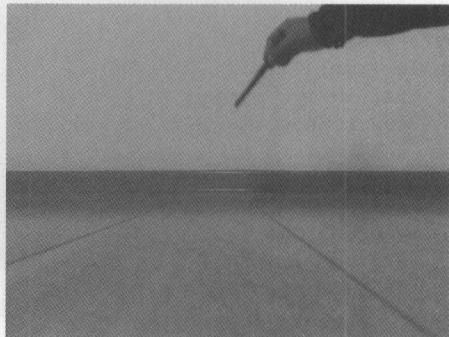


图 11-17 视频的截图 4





图 11-18 视频的截图 5

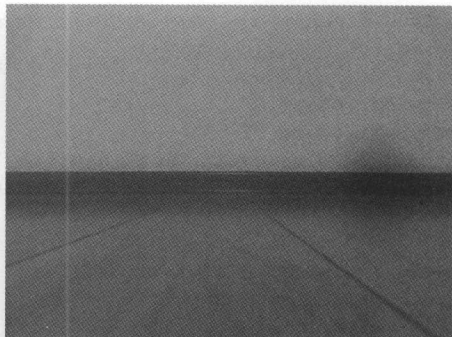


图 11-19 视频的截图 6

## 2. 基于相邻帧差的算法

基于相邻帧差的算法过程如下：

1) 将当前帧的灰度图像与前一帧的灰度图像相减，得到差分值，当差分值大于某个阈值时，将差分矩阵的相应位置设置为 1，否则设置为 0，差分矩阵的计算公式如下：

$$D(i, j) = \begin{cases} 1 & f_{k-1}(i, j) - f_k(i, j) > THRESHOLD \\ 0 & \text{其他} \end{cases}$$

其中， $f_k(i, j)$  表示第  $k$  个图像矩阵  $(i, j)$  处的灰度值， $D(i, j)$  表示差分矩阵  $(i, j)$  处的值。

2) 如果差分矩阵不全为 0，则表示检测到运动。如果有必要，可按一定的运动检测次数对运动画面进行分组，并融合叠加形成多组运动轨迹图。

3) 取下一帧图像，转到 (1)，直到所有帧图像全部处理完毕后退。

用 Python 代码实现该算法，如程序 11-7.py 所示。

```

# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 差分运动检测，相邻帧差检测画面中是否有运动的物体
#11-7.py
import cv2
import numpy as np
# 读取运动序列图像
fn=[]
for i in xrange(6):
    fn.append(str(i+1)+".png")

img=[]
colorimg=[]
myimg=[]

for i in xrange(6):
    tmpimg=cv2.imread(fn[i])
    colorimg.append(tmpimg)
    myimg=cv2.cvtColor(tmpimg,cv2.COLOR_BGR2GRAY)
    img.append(myimg)

```

```

# 差分计算
myimg1=colorimg[0].copy()
myimg2=myimg1.copy()
w=myimg1.shape[1]
h=myimg1.shape[0]
moveimg1=np.zeros((h,w),np.uint8 )
moveimg2=np.zeros((h,w,3),np.uint8 )
for ii in xrange(5):

    print u" 开始分析第 "+str(ii+2)+u" 个运动图像 ..."
    myd=img[ii+1]-img[ii]
    # 生成差分矩阵
    THRESHOLD=int(np.median(abs(myd)))# 取中位数做为阈值
    mymove=np.ones([h,w],np.uint8)
    for i in xrange(h):
        for j in xrange(w):
            if abs(myd[i,j])<THRESHOLD or myd[i,j]==0:
                mymove[i,j]=0
    # 如果有物体运动则输出报警, 并生成运动效果叠加图
    if np.sum(mymove)>0 :
        print u" 第 "+str(ii+2)+u" 个运动图像发生了变化 !"
        moveimg1=img[ii+1]*(1-mymove)*0.16+img[ii]*(1-mymove)*0.16+moveimg1
        moveimg2=colorimg[ii+1]*0.16+colorimg[ii]*0.16+moveimg2
        moveimg1=np.array(moveimg1,np.uint8)
        moveimg2=np.array(moveimg2,np.uint8)

# 显示运动画面
# 计算运动部分
moveimg1=moveimg1-img[0]
# 灰度图二值化以更好地显示运动的效果。
retval, showing1=cv2.threshold(moveimg1,140,255,cv2.THRESH_BINARY)
showimg1=showimg1
showimg2=moveimg2
cv2.imshow("move1",showimg1)
cv2.imshow("move2",showimg2)
cv2.waitKey()
cv2.destroyAllWindows()

```

运行程序 11-7.py, 程序检测到第 2、3、4、6 帧时, 图像发生了变化, 运行结果如下:

```

开始分析第 2 个运动图像 ...
第 2 个运动图像发生了变化 !
开始分析第 3 个运动图像 ...
第 3 个运动图像发生了变化 !
开始分析第 4 个运动图像 ...
第 4 个运动图像发生了变化 !
开始分析第 5 个运动图像 ...
开始分析第 6 个运动图像 ...
第 6 个运动图像发生了变化 !

```

程序 11-7.py 绘制了使用相邻帧差分法检测的运动效果, 如图 11-20 所示。

观察图 11-20, 左边的图像为运动图像的融合, 右边的图像为程序提取运动部分的灰度图, 可较直观地看出手拿着笔摆动时的运动路径和运动速度。

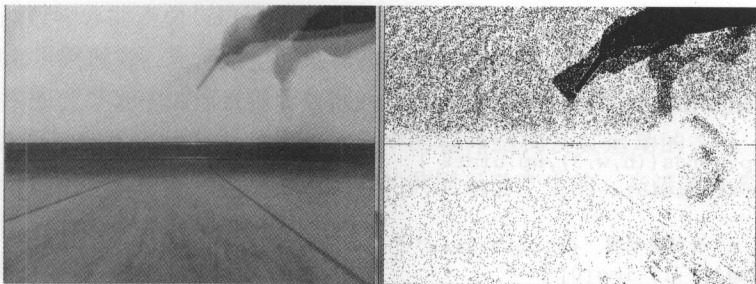


图 11-20 运动效果

### 3. 基于背景图像与当前帧差的算法

该算法的具体过程如下：

1) 将当前帧的灰度图像与背景灰度图像相减，得到差分值，当差分值大于某个阈值时，将差分矩阵的相应位置设置为 1，否则设置为 0，差分矩阵计算公式如下：

$$D(i, j) = \begin{cases} 1 & f_{k-1}(i, j) - f_i(i, j) > THRESHOLD \quad \text{其中 } k \in [2, n] \\ 0 & \text{其他} \end{cases}$$

上式中， $f_k(i, j)$  表示第  $k$  个图像矩阵  $(i, j)$  处的灰度值， $D(i, j)$  表示差分矩阵  $(i, j)$  处的值。

2) 如果差分矩阵不全为 0，则表示检测到运动。如果有必要，可按一定的运动检测次数对运动画面进行分组，并融合叠加形成多组运动轨迹图。

3) 取下一帧图像，转到 (1)，直到  $n$  个帧的图像全部处理完毕后退出。

用 Python 代码实现该算法，如程序 11-8.py 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 差分运动检测，基于背景图像与当前帧差来检测画面中是否有运动的物体
#11-8.py
import cv2
import numpy as np

# 读取运动序列图像
fn=[]
for i in xrange(6):
    fn.append("mv"+str(i+1)+".png")

img=[]
colorimg=[]
myimg=[]

for i in xrange(6):
    tmpimg=(cv2.imread(fn[i]))
    colorimg.append(tmpimg)
    myimg=cv2.cvtColor(tmpimg,cv2.COLOR_BGR2GRAY)
    img.append(myimg)
```

```

# 差分计算
myimg=colorimg[0].copy()
w=myimg.shape[1]
h=myimg.shape[0]

moveimg=np.zeros((h,w,3),np.uint8 )
for ii in xrange(5):

    print u" 开始分析第 "+str(ii+2)+u" 个运动图像 ..."
    myd=img[ii+1]-img[0]
    # 生成差分矩阵
    THRESHOLD=int(np.median(abs(myd)))# 取中位数做为阈值
    mymove=np.ones([h,w],np.uint8)
    for i in xrange(h):
        for j in xrange(w):
            if abs(myd[i,j])<THRESHOLD or myd[i,j]==0:
                mymove[i,j]=0
    # 如果有物体运动则输出报警
    if np.sum(mymove)>0 :
        print u" 第 "+str(ii+2)+u" 个运动图像发生了变化 !"
        moveimg=colorimg[ii+1]*0.16+colorimg[ii]*0.16+moveimg
        moveimg=np.array(moveimg,np.uint8)

# 显示移动部分
showimg=moveimg
cv2.imshow("move",showimg)
cv2.waitKey()
cv2.destroyAllWindows()

```

程序 11-8.py 绘制了检测到的运动效果 (如图 11-21 所示), 并输出结果如下:

```

开始分析第 2 个运动图像 ...
第 2 个运动图像发生了变化 !
开始分析第 3 个运动图像 ...
第 3 个运动图像发生了变化 !
开始分析第 4 个运动图像 ...
第 4 个运动图像发生了变化 !
开始分析第 5 个运动图像 ...
第 5 个运动图像发生了变化 !
开始分析第 6 个运动图像 ...

```

从图 11-21 中可较直观地看出手拿着笔摆动时的运动路径和运动速度。

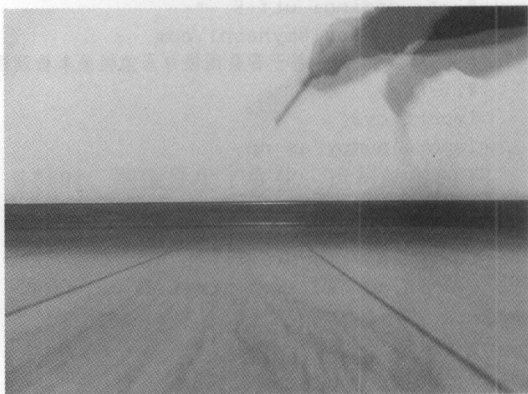


图 11-21 检测到的运动

### 11.3.3 光流法

#### 1. 光流法概述

光流 (Optical Flow/Optic Flow) 是关于视域中的物体运动检测的概念。用来描述相对于观察者的运动所造成的观测目标、表面或边缘的运动。光流法在模式识别、计算机视觉及其他图像处理中非常有用, 可用于运动检测、物件切割、碰撞时间与物体膨胀的计算、运动补



偿编码, 或者通过物体的表面与边缘进行立体的测量等。

光流法的基本原理为: 光流是空间运动物体在观测成像面上的像素运动的瞬时速度。光流的研究是利用图像序列中的像素强度数据的时域变化和相关性来确定各自像素位置的“运动”, 即研究图像灰度在时间上的变化与景象中的物体结构及其运动的关系。一般情况下, 光流是由相机运动、场景中的目标运动或两者的共同运动产生的。光流的计算方法大致可分为三类: 基于匹配的、频域的和梯度的方法。

光流法的前提假设如下:

- 相邻帧之间的亮度恒定。
- 相邻视频帧的取帧时间是连续的, 或者说, 相邻帧之间物体的运动比较“微小”。
- 保持空间的一致性, 即同一子图像的像素点具有相同的运动。

## 2. Python 实现

程序 11-9.py 从摄像头采集图像, 然后对每个连续的图像进行光流估计, 并绘制流矢量, 效果如图 11-22 所示。

```
# -*- coding: utf-8 -*-
#coding:myhaspl@myhaspl.com
# 光流法
#11-9.py
import cv2
import numpy as np

def flowdraw(im,flow,step=14):
    # 绘制光流
    h,w=im.shape[:2]
    y,x=np.mgrid[step/2:h:step,step/2:w:step].reshape(2,-1)
    fx,fy=flow[y,x].T
    # 线终点
    lines=np.vstack([x,y,x+fx,y+fy]).T.reshape(-1,2,2)
    lines=np.int32(lines)
    # 创建图像
    myvis=cv2.cvtColor(im,cv2.COLOR_GRAY2BGR)
    for (x1,y1),(x2,y2) in lines:
        cv2.line(myvis,(x1,y1),(x2,y2),(0,255,0),1)
        cv2.circle(myvis,(x1,y1),1,(0,255,0),-1)
    return myvis
# 设置需要采集视频的设备 ID 为 0, 从第 0 个摄像头采集
mycap=cv2.VideoCapture(0)
# 前一个图像
ret,im=mycap.read()
prev_pic=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

while True:
    ret,im=mycap.read()
    # 采集
    pic=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    # 计算流
    myflow=cv2.calcOpticalFlowFarneback(prev_pic,pic,0.5,3,15,3,5,1,0)
```



```

# 上帧图像赋值
prev_pic=pic
# 绘制流矢量
cv2.imshow('myvideo flow ',flowdraw(pic,myflow))
# 每 15 毫秒采集 1 次
key=cv2.waitKey(15)
# 空格键退出
if key==32:
    break

```

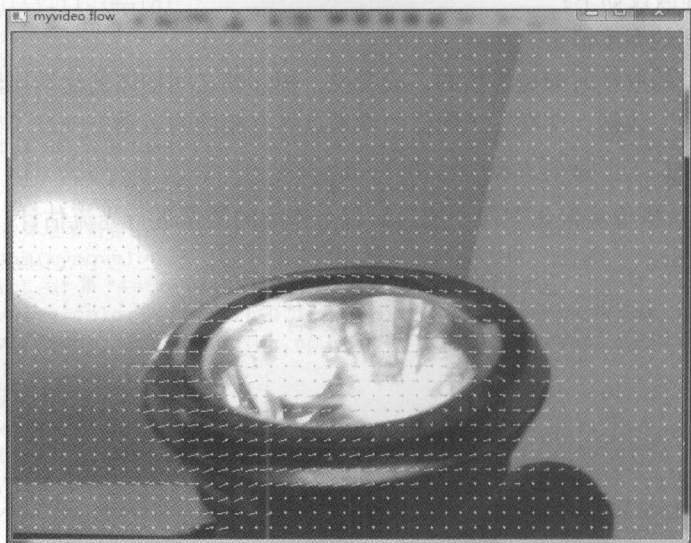


图 11-22 光流法

观察图 11-22，从流矢量线条及箭头的方向中可以发现手电筒正在从左到右移动。

## 11.4 形状检测

### 11.4.1 KNN 算法概述

K 最近邻 (KNN, k-NearestNeighbor) 分类算法可以说是整个数据分类技术中最简单的方法了。所谓 K 最近邻，就是  $k$  个最近的邻居的意思，说的是每个样本都可以用它最靠近的  $k$  个邻居来代表。KNN 算法的核心思想是：如果一个样本在特征空间中的  $k$  个最相邻的样本中的大多数都属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性，算法在确定分类决策上只需要依据最邻近的一个或几个样本的类别即可决定待分样本所属的类别，因此，在做类别决策时，只与极少量的相邻样本有关。

由于 KNN 方法主要靠周围有限的邻近样本，而不是靠判别类域的方法来确定所属的类别，因此对于类域的交叉或重叠较多的待分样本集来说，KNN 方法较其他的方法更为合适。

以二维样本为例，利用 KNN 方法将样本分成两类。图 11-23 中三角形和方形是已知类别的样本点，这里假设三角形为正类，方形为负类，圆形点是未知类别的数据，现在需要利用这些已知类别的样本对圆形点进行分类。

对图 11-23 的圆形点进行 KNN 分类的具体过程如下：

1) 事先定下  $k$  值（就是指 K 近邻方法中  $k$  的大小，代表对于一个待分类的数据点，要寻找它的几个邻居）。为讲解方便，取两个  $k$  值，分别为 3 和 5。

2) 根据事先确定的距离度量公式（如：欧氏距离），得出待分类数据点和所有已知类别的样本点中，距离最近的  $k$  个样本。

3) 统计这  $k$  个样本点中各个类别的数量。如图 12-15，如果选定  $k$  值为 3，则正类样本（三角形）有 2 个，负类样本（方形）有 1 个，那

么就把这个圆形数据点定为正类；而如果选择  $k$  值为 5，则正类样本（三角形）有 2 个，负类样本（方形）有 3 个，那么将这个数据点定为负类。即，根据  $k$  个样本中，数量最多的样本是什么类别，就把这个数据点定为什么类别。

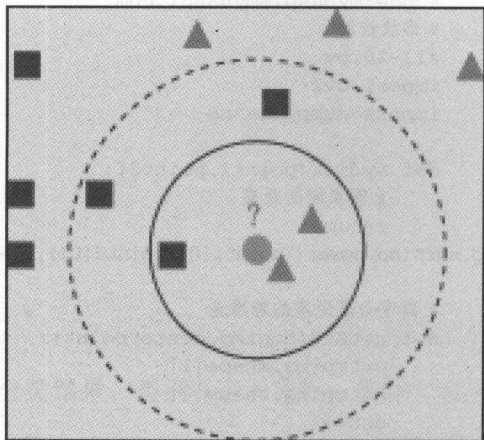


图 11-23 K 近邻算法示意图

### 11.4.2 形状特征提取

图 11-24 是三个典型的形状，分别是方形、圆形和三角形。

那么如何提取图 11-24 所示的三个形状的特征呢，可使用距中心距离法，具体过程如下：

1) 确定图像的中心点 center，通常是图片的中间位置。

2) 计算图像线条上的每个点  $x$  与中心点 center 之间的距离（两点间的距离公式），形成距离向量  $d$ 。

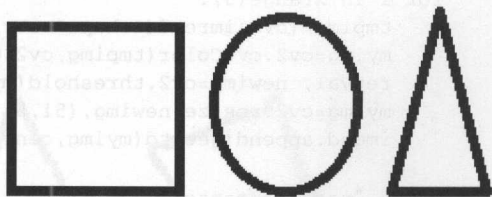


图 11-24 典型的形状

3) 计算距离向量  $d$  的标准差。以标准差为最终特征值。

### 11.4.3 形状分类

本算法为 KNN 算法的变体，本案例每种形状只提供了一个样本，因此，将 KNN 算法中的  $k$  值取 1，以最近的那个邻近点的类别判断未知形状的分类。

算法具体过程为：首先采集样本图像（图 11-24 所示的样本图像文件名从左到右分别是 shape0.png、shape1.png、shape2.png）的特征值，然后以未知形状图像的特征值为分类数据，进行 KNN 分类，最后将未知图像归为上图所示的 3 类之中，从左边数起，类别分别为 0（方

形)、1 (圆形)、2 (三角形), Python 代码如程序 11-10.py 所示。

```
# -*- coding: utf-8 -*-
#code:myhaspl@myhaspl.com
# 形状检测
#11-10.py
import cv2
import numpy as np

def mydist(point1,point2):
    # 两点间的距离
    return
np.sqrt(np.power((point1[0]-point2[0]),2)+np.power((point1[1]-point2[1]),2))

# 离中心点距离的标准差
def getstd(tmpimg,centerpoint):
    w=tmpimg.shape[1]
    h=tmpimg.shape[0]
    dst=[]
    for i in xrange(h):
        for j in xrange(w):
            if tmpimg[i,j]<255:
                # 该像素不是空白,是线条中的点
                dst.append(mydist((i,j),centerpoint))
    mysd=np.std(dst)
    return mysd
# 读取样本形状图像,计算标准差
imgsd=[]
testimgsd=[]
mysdratio=[]
centerpoint=(26,26)
for i in xrange(3):
    tmpimg=(cv2.imread("shape"+str(i)+".png"))
    myimg=cv2.cvtColor(tmpimg,cv2.COLOR_BGR2GRAY)
    retval, newimg=cv2.threshold(myimg,40,255,cv2.THRESH_BINARY)
    myimg=cv2.resize(newimg,(51,51))
    imgsd.append(getstd(myimg,centerpoint))

print "-----"

for i in xrange(4):
    # 测试样本
    fn="shapetest_"+str(i)+".png"
    tmpimg=(cv2.imread(fn))
    myimg=cv2.cvtColor(tmpimg,cv2.COLOR_BGR2GRAY)
    retval, newimg=cv2.threshold(myimg,40,255,cv2.THRESH_BINARY)
    myimg=cv2.resize(newimg,(51,51))
    # 得到距离标准差
    testimgsd.append(getstd(myimg,centerpoint))
    mysdratio.append([])
    for ii in xrange(3):
        # 与样本进行比较
```

```

mysdratio[i].append(abs(testimgsd[i]-imgsd[ii]))

print u"KNN 距离矩阵如下 "
print mysdratio
print "-----"
#KNN 算法, 找到最近邻
for i in xrange(4):
    fn="shapetest_"+str(i)+".png"
    myindex=0
    mymin=np.min(mysdratio[i])
    for ii in xrange(3):
        # 测试样本最终分类
        if (mysdratio[i][ii])==mymin:
            myindex=ii
            break
    print fn+u" 分类为: "+str(myindex)

```

执行程序 11-10.py 后, 输出 KNN 距离矩阵及分类结果, 类别 0 表示方形, 类别 1 表示圆形, 类别 2 表示三角形。结果如下:

```

-----
KNN 距离矩阵如下
[[0.71030223330565079, 0.75471514364899916, 3.9533561391620591],
 [1.2922713309313281, 0.17274604602332189, 4.5353252367877364],
 [2.0123227209027084, 3.4773400978573585, 1.2307311849536999],
 [0.49271253133816462, 0.97230484561648534, 3.7357664371945729]]
-----
shapetest_0.png 分类为: 0
shapetest_1.png 分类为: 1
shapetest_2.png 分类为: 2
shapetest_3.png 分类为: 0

```

程序 11-10.py 使用的测试图像如图 11-25 所示。

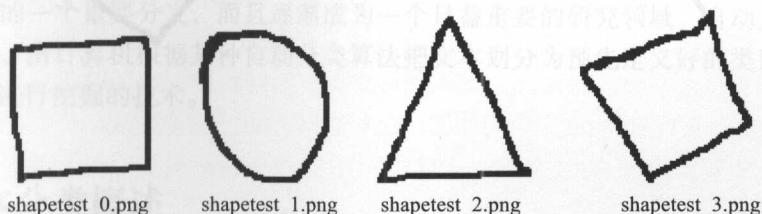


图 11-25 测试图像

观察图 11-25 及程序 11-10.py 的输出结果, 可发现测试图像的分类结果是正确的, 效果还不错。

为提高分类准确率, 实践中可采用标准的 KNN 算法, 为每种形状提供更多的样本, 分类的方法为: 如果与未知形状的  $k$  个最相似的大多数样本属于某一个类别, 那么该未知形状也属于这个类别。



## 11.5 小结

计算机视觉研究使机器能看懂图像，属于人工智能和机器学习的范畴。目前机器视觉实现的主要方式是：首先，使用摄像机等设备采集画面，生成数字图像；然后，对图像进行预处理和加工，提取特征；最后，分析图像特征，挖掘图像包含的知识和信息。本章讲述了运用机器学习算法对视频和图像进行加工、分析、提取特征、总结知识等技术，首先介绍了人脸辨识的算法，然后讲解了手写数字识别的算法，最后讲述了运动侦测技术和形状检测的方法，在讲解这些算法的过程中，注重实践效果，每个实例均用 Python 进行实现，以验证算法的有效性。

## 思考题

- (1) 以多个图像进行更多的人脸辨识实验，尝试应用更多的机器学习算法，比如应用 SVM 和神经网络，或者对欧氏距离算法进行进一步改进，对人脸的不同部位赋予不同的权重，总结各算法的实际应用效果。
- (2) 实现从 a 到 z 的 24 个字母图像的识别算法，要求识别未知样本图像表征的字母。
- (3) 运用本章介绍的运动侦测方法，分析自己的计算机摄像头所采集的视频。
- (4) 运用本章介绍的形状检测方法或其他方法，对某类复杂的鼠标（或触摸屏输入）手势进行识别，比如以下手势：

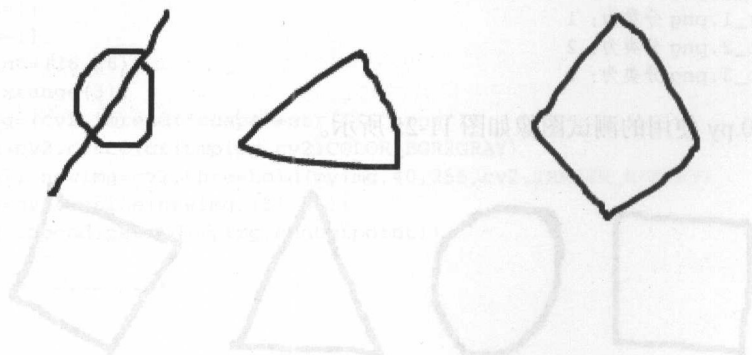


图 11-25 图形识别





## 第12章 Chapter 12

# 文本分类案例

在网络信息时代，全球正面临着前所未有的信息爆炸式增长的挑战，中文信息处理也迎来了高速发展。如果拥有着海量的中文文字数据对数据知识的提取依旧停留在过去简单查询、检索的水平上，那么就会导致所谓的“数据爆炸但知识贫乏”的现象。从中文信息中获取知识，快速有效地组织和管理用户所要的信息是当前信息科学和技术所面临的重要挑战。

数据挖掘技术致力于从海量数据中提取有用的信息，文本分类等文本挖掘技术则属于数据挖掘领域中被研究的热点。文本挖掘是人工智能、机器学习、自然语言处理、数据挖掘及相关自动文本处理（如信息抽取、信息检索）等领域中理论和技术结合的产物。文本分类是文本挖掘领域的一个重要分支，而且逐渐成为一个日益重要的研究领域。自动文本分类则会依据文本内容，由计算机根据某种自动分类算法把文本划分为预先定义好的类别，它是对复杂类型的数据进行挖掘的技术。

## 12.1 文本分类概述

近年来，随着互联网的高速发展和大数据时代的到来，文本分类等文本挖掘技术应用于越来越多的领域。互联网和大数据是信息技术发展催生的一对孪生子。互联网能够方便、准确地记录用户数据，产生了大量的半结构化、非结构化的文本数据，这也使互联网成为大数据分析应用最广泛的领域之一。例如：搜索引擎技术的基础就是基于文本分析算法的，而面向互联网用户的精准营销则是以广告数据分析（现在已经催生了一门科学“计算广告学”）为依据的。海量文档数据库需要高效的文本挖掘算法作为数据索引、查询分析的核心算法，智

能输入法需要分析用户输入习惯及输入的词句,自动客服系统由原始的词语匹配技术转变为基于文本挖掘的自然语言理解算法。

分类技术是数据挖掘中非常重要的分支。分类就是根据数据集的特点找出类别的概念描述,这个概念描述代表了这类数据的整体信息,也就是该类的内涵描述,使用这种类的描述可对未来的测试数据进行分类。

文本挖掘算法在搜索引擎中一直扮演着重要的角色,搜索引擎每个阶段的发展都伴随着文本挖掘技术的进步。1990年,作为可搜索FTP文件名列表的Archie,通过文件名匹配技术完成了文本检索。1993年,世界上第一个Spider程序——World Wide Web Wanderer开始在网络间爬取资料,统计互联网上的服务器数量。同年,Stanford大学的学生创造了Excite,它通过分析字词关系进行更有效的检索,文本挖掘技术再次进步。1994年,杨致远和David Filo共创办了Yahoo, Yahoo网站的分类目录由人工整理维护,他们会精选互联网上的优秀网站,进行简要描述后,分类放置到不同目录下。用户查询时,通过一层层的点击来查找自己想找的网站,人工分类精准度高,但工作量很大。同年Lycos正式发布, Lycos使用了更先进的文本挖掘技术,包括:相关性排序、前缀匹配和字符相近限制、网页自动摘要等。1995年,更多的文本挖掘技术陆续产生, AltaVista在搜索引擎中引入了自然语言搜索技术,实现高级搜索语法(如AND、OR、NOT等)。1997年,文本自动分类技术首次应用于Northernlight搜索引擎,该搜索引擎支持对搜索结果进行简单的自动分类。同年, google.com的域名被Larry Page注册, Google横空出世,之后发展成为今天的搜索业巨头。2013年全球在线广告营收中, Google预计占有超过33%的市场份额,在全球移动广告营收中, Google的市场份额约56%。Google研究和应用了大量文本挖掘算法,例如:Pagerank、动态摘要、网页快照、DailyRefresh、多文档格式支持等。有人评论说Google永远改变了搜索引擎的定义。2001年,百度搜索引擎发布,和Google一样,百度也研发和使用了很多文本挖掘算法:网页快照、相关搜索词、错别字纠正、Flash搜索、信息快递搜索、图像搜索、新闻搜索等。

近年来,借助模式识别算法,文本分类技术飞速发展。文本分类大致分为几个要素:文本向量模型表示、文本特征选择和文本训练分类器。目前比较流行的分类方法主要有SVM、改进余弦相似度、贝叶斯方法、神经网络、k2最近邻方法、遗传算法、粗糙集等。文本分类算法通常包括文本预处理(中文分词、去除停用词)、文本特征提取、样本特征学习及算法对未知样本的预测等过程。

本章将以余弦相似度、朴素贝叶斯分类算法为主,阐述文本分类技术。

## 12.2 余弦相似度分类

基于余弦相似度的文本分类算法实现的基本过程为:首先对样本文本进行分词,接着将垃圾词剔除,然后根据剔除后的词条把样本文本中的所有词映射到 $n$ 维空间的一个向量上,并计算未知文本特征组形成的向量与各类别特征组向量之间夹角的余弦值,最后通过比较余

弦值的大小判断最接近的分类。

## 12.2.1 中文分词

中文分词指的是将一个汉字序列切分成一个个单独的词。中文分词是文本挖掘的基础，对于一段中文文本，中文分词是文本自动识别的前提。目前常用的中文分词软件主要有以下几种：

- SCWS。基于词频词典的机械中文分词引擎，它能将一整段的汉字基本正确地切分成词。采用的是采集的词频词典（词频 TF 指某一个给定的词语在一份给定的文件里出现的次数，近年来，利用计算机进行词频统计，以词频统计的结果来验证词典收词的得失，决定哪些词需要收录，中文词频词典按词频排序存储词语和词组），并辅以一定的专有名称、人名、地名、数字、年代等识别规则，从而达到基本分词的目的。
- ICTCLAS。这是最早的中文开源分词项目，在国内 973 专家组组织的评测活动中获得了第一名，在第一届国际中文处理研究机构 SigHan 组织的评测中获得了多项第一名。ICTCLAS 全部采用 C/C++ 编写，支持 Linux、FreeBSD 及 Windows 系列操作系统，支持 C/C++、C#、Delphi、Java 等主流的开发语言。
- HTTPCWS。基于 HTTP 协议的开源中文分词系统，将取代之前的 PHPCWS 中文分词扩展。
- 庖丁解牛分词。仅支持 Java 语言，且提供 lucence（一款流行的 Java 全文搜索引擎）接口。
- CC-CEDICT。提供一份以汉语拼音为中文辅助的汉英辞典，其词典可以用于中文分词，Chrome 中文版就是使用这个词典进行中文分词的。
- “结巴”（Jieba）中文分词。Python 中文分词组件 Jieba 支持 3 种分词模式：精确模式，试图将句子最精确地切开，适合文本分析；全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

本章基于 Python 实现算法，因此选择“结巴”中文分词作为分词组件库。可在 <http://pypi.python.org/pypi/jieba> 处下载，解压后运行 `pythonsetup.py install` 进行安装。下面的代码演示了分词组件 Jieba 的基本使用方法。

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#12-1.py
import sys
sys.path.append("../")
import jieba

seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print "Full Mode:", "/ ".join(seg_list) # 全模式

seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
```

```
print "Default Mode:", "/" .join(seg_list) # 默认模式

seg_list = jieba.cut("他来到了网易杭研大厦")
print "/", ".join(seg_list)

seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造") #
搜索引擎模式
print "/", ".join(seg_list)
```

演示程序的分词效果如下：

```
/ 我 / 来到 / 北京 / 清华 / 清华大学 / 华大 / 大学 /
Default Mode: 我 / 来到 / 北京 / 清华大学
他, 来到, 了, 网易, 杭研, 大厦
小明, 硕士, 毕业, 于, 中国, 科学, 学院, 科学院, 中国科学院, 计算, 计算所, , , 后,
在, 日本, 京都, 大学, 日本京都大学, 深造
```

中文分词有一个困难，就是会遇到歧义。同样的一句话，可能有两种或者更多的切分方法。主要的歧义有两种：交集体歧义和组合型歧义。例如：语句中出现了“漂亮的”，因为“漂亮”和“亮的”都是词，那么这个短语就可以分成“漂亮/的”和“漂/亮的”，这种称为交集体歧义。组合型歧义情况更复杂，要根据整个句型来判断，比如：句子“2010年底部队友谊篮球赛结束”，“底部”是一个词，“年底”是一个词，“部队”是一个词，“队友”是一个词，“友谊”是一个词，而分词不能曲解句子含义，这样就产生了分词困难。

目前大部分分词软件能较好地解决歧义问题。下面试着应用“结巴”分词对“2010年底部队友谊篮球赛结束”分词。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#12-2.py
import jieba
seg_list = jieba.cut("2010年底部队友谊篮球赛结束", cut_all=False)
print "Default Mode:", "/" .join(seg_list) # 默认模式
```

从下面的执行结果来看，分词效果很理想。“结巴”分词技术比较成熟，能应用于实际工程中。

```
Default Mode:Building Trie..., from E:\WinPython-32bit-2.7.5.1\python-2.7.5\
lib\site-packages\jieba\dict.txt
loading model from cache c:\users\admini~1\appdata\local\temp\jieba.cache
2010/ 年底 / 部队 / 友谊 / 篮球赛 / 结束
loading model cost 1.26200008392 seconds.
Trie has been built succesfully.
```

上面执行结果的第一行中的 Trie 是一种数据结构，“结巴”分词使用它构造词条字典。Trie 称前缀树或字典树，是一种有序树，用于保存关联数组，其中的键通常是字符串。与二叉查找树不同，键不是直接保存在节点中的，而是由节点在树中的位置决定的。一个节点的所有子孙都有相同的前缀，也就是这个节点对应的字符串，而根节点对应空字符串。



## 12.2.2 停用词清理

停用词又称垃圾词。完成自然语言理解与文本分类等任务时，都需要预处理文本，自动过滤掉某些不能表征意义的字、词或符号，这些字或词被称为停用词（Stop Words）。进行文本分词后形成的词条组中会存在很多停用词，这些词基本不具备表示文本特征的能力，其存在会影响其他词对文本特征的表征能力，因此，需要过滤后才能形成“干净”的文本特征码。

例如：对下面这句话进行分词：“春秋时期有一个农夫，他总是嫌田里的庄稼长得太慢，今天去瞧瞧，明天去看看，觉得禾苗好像总没有长高。他心想：有什么办法能使它们长得高些快些呢？”代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#12-3.py

import jieba
seg_list = jieba.cut(" 春秋时期有一个农夫，他总是嫌田里的庄稼长得太慢，今天去瞧瞧，明天去看，觉得禾苗好像总没有长高。他心想：有什么办法能使它们长得高些快些呢？ ", cut_all=False)
print "Default Mode:", "/ ".join(seg_list)## 默认模式
.....
.....
```

分词效果如下：

```
春秋时期 / 有 / 一个 / 农夫 / ， / 他 / 总是 / 嫌 / 田里 / 的 / 庄稼 / 长得 / 太慢 / ， / 今天 / 去 / 瞧瞧 / ， / 明天 / 去 / 看看 / ， / 觉得 / 禾苗 / 好像 / 总 / 没有 / 长高 / 。 / 他 / 心想 / ： / 有 / 什么 / 办法 / 能 / 使 / 它们 / 长得 / 高些 / 快些 / 呢 / ？ /
```

上述分词结果中，“有”“能”“呢”“什么”等词以及标点符号都属于停用词的范围，应对其进行清理。清理的方式是建立停用词表，扫描分词结果，从中剔除停用词表中的词条。代码如下：

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#12-3.py

import jieba
seg_list = jieba.cut(" 春秋时期有一个农夫，他总是嫌田里的庄稼长得太慢，今天去瞧瞧，明天去看，觉得禾苗好像总没有长高。他心想：有什么办法能使它们长得高些快些呢？ ", cut_all=False)
liststr="/ ".join(seg_list)
print u"----- 清理前的词条 -----"
print "Default Mode:", liststr## 默认模式
print u"----- 清理后的词条 -----"
# 停用词清理
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read( )
    f_stop_text=unicode(f_stop_text,'utf-8')
```



```
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')
for myword in liststr.split('/'):
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        print myword,',',
```

从直观上看，清理停用词后留下的文本词条表征文本的能力都比较强。来比较一下：

----- 清理前的词条 -----

Default Mode: 春秋时期 / 有 / 一个 / 农夫 / , / 他 / 总是 / 嫌 / 田里 / 的 / 庄稼 / 长得 / 太慢 / , / 今天 / 去 / 瞧瞧 / , / 明天 / 去 / 看看 / , / 觉得 / 禾苗 / 好像 / 总 / 没有 / 长高 / 。 / 他 / 心想 : / 有 / 什么 / 办法 / 能 / 使 / 它们 / 长得 / 高些 / 快些 / 呢 / ? /

----- 清理后的词条 -----

春秋时期 , 一个 , 农夫 , 总是 , 田里 , 庄稼 , 长得 , 太慢 , 今天 , 瞧瞧 , 明天 , 看看 , 觉得 , 禾苗 , 好像 , 长高 , 心想 , 办法 , 长得 , 高些 , 快些 ,

## 12.2.3 算法实战

### 1. 任务描述

假设提供了一个描述战争的样本数据，如图 12-1 所示。现在需要对两个未知类型文本进行分析，它们的内容如图 12-2 和图 12-3 所示，判断哪个文本是描述战争类的。

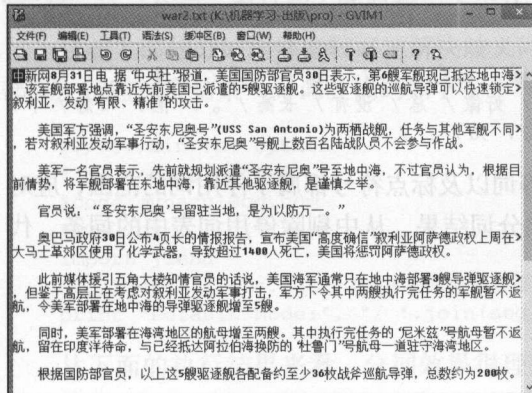


图 12-1 描述战争的样本文本

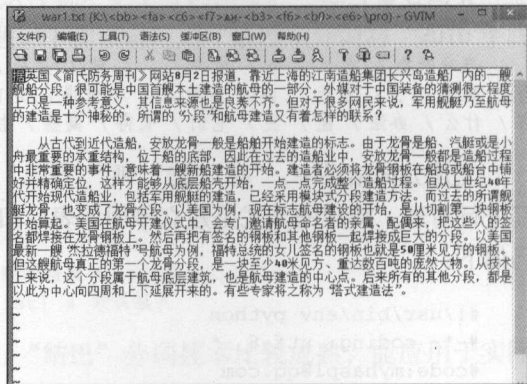


图 12-2 描述战争的未知类型文本

### 2. 算法过程

用余弦相似度算法完成上述文本分类任务的过程如下：

- 1) 读取样本文本。
- 2) 对文本进行 utf-8 编码转换。
- 3) 对文本进行预处理，完成中文分词，形成词条库，并去除停用词。
- 4) 读取文本词条库，统计每个词条的词频。词频代表了每个词对一段文本的重要程度，字词的重要性随着它在文件中出现的次数成正比增加。
- 5) 将上一步整理形成的每个词的词频组成文本的词条词频特征码。

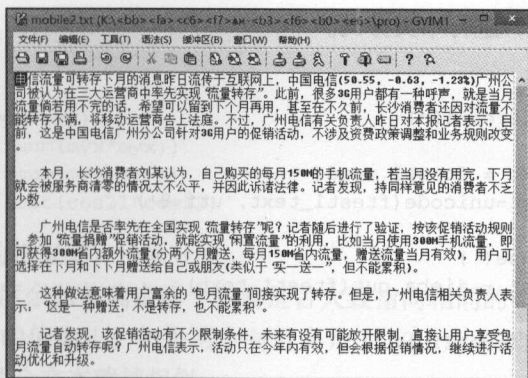


图 12-3 描述手机的未知类型文本

6) 使用前面第1步到第5步的方法分析待分类文本，生成待分类文本的词条词频特征码。

7) 将待分类文本的词条词频特征码与样本的词条词频特征码进行比较，应用余弦相似度算法判断待分类文本与样本的相似度，取最相似的类型为最终分类的类型。

下面用 Python 实现上述算法过程。

1) 读取样本文本，完成 utf-8 编码转换，然后进行中文分词。

```
print
print 'loading ...'
print 'working',
f1 = open(sampfn)
try:
    f1_text = f1.read()
    f1_text=unicode(f1_text,'utf-8')
finally:
    f1.close()
f1_seg_list = jieba.cut(f1_text)
```

2) 对文本词条进行预处理，去除停用词，计算每个词条的词频。

# 去除停用词，同时构造样本词的字典

```
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

test_words={}
all_words={}
for myword in f1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        test_words.setdefault(myword,0)
        all_words.setdefault(myword,0)
        all_words[myword]+=1
```

## 3) 读取待分类文本, 进行中文分词。

```
# 第一个待测试数据
ftest1 = open(ftest1fn)
try:
    ftest1_text = ftest1.read()
    ftest1_text=unicode(ftest1_text,'utf-8')
finally:
    ftest1.close()
ftest1_seg_list = jieba.cut(ftest1_text)
# 第二个待测试数据
ftest2 = open(ftest2fn)
try:
    ftest2_text = ftest2.read()
ftest2_text=unicode(ftest2_text,'utf-8')
finally:
    ftest2.close()
ftest2_seg_list = jieba.cut(ftest2_text)
```

## 4) 继续预处理待分类文本, 去除停用词, 并生成词频特征码。

```
# 读取待测试文本
mytest1_words=copy.deepcopy(test_words)
for myword in ftest1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest1_words.has_key(myword):
            mytest1_words[myword]+=1

mytest2_words=copy.deepcopy(test_words)
for myword in ftest2_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest2_words.has_key(myword):
            mytest2_words[myword]+=1
```

## 5) 计算并输出样本与待测试文本的余弦相似度。

```
# 计算样本与待测试文本的余弦相似度
sampdata=[]
test1data=[]
test2data=[]
for key in all_words.keys():
    sampdata.append(all_words[key])
    test1data.append(mytest1_words[key])
    test2data.append(mytest2_words[key])
test1simi=get_cossimi(sampdata,test1data)
test2simi=get_cossimi(sampdata,test2data)
print u"%s 与样本 [%s] 的余弦相似度:%f"%(ftest1fn,sampfn,test1simi)
print u"%s 与样本 [%s] 的余弦相似度:%f"%(ftest2fn,sampfn,test2simi)
```

上面这段代码调用了 `get_cossimi` 函数, 这是余弦相似度计算函数。该函数的定义如下:

```
def get_cossimi(x,y):
    myx=np.array(x)
    myy=np.array(y)
    cos1=np.sum(myx*myy)
    cos21=np.sqrt(sum(myx*myx))
    cos22=np.sqrt(sum(myy*myy))
    return cos1/float(cos21*cos22)
```

## 6) 根据屏幕输出的相似度, 预测分类。

两个向量之间的角度余弦值确定两个向量是否大致指向相同的方向。如果两个向量有相同的指向, 余弦相似度的值为 1; 如果两个向量夹角为 90, 余弦相似度的值则为 0。可见, 余弦相似度越接近 1, 两个文本就越相似。

```
.....
mobile2.txt 与样本 [war2.txt] 的余弦相似度:0.160806
war1.txt 与样本 [war2.txt] 的余弦相似度:0.264215
```

上面是代码的执行结果, 分析这个结果可得出结论: mobile2.txt 与 war2.txt 的余弦相似度较小, war1.txt 与 war2.txt 的余弦相似度为 0.264215, 更接近 1。因此, 应将 war1.txt 文本文件划分为战争类。

以下是全部源代码:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#12-4.py
```

```
import numpy as np
import jieba
import copy
ftest1fn='mobile2.txt'
ftest2fn='war1.txt'
sampfn='war2.txt'
```

```
def get_cossimi(x,y):
    myx=np.array(x)
    myy=np.array(y)
    cos1=np.sum(myx*myy)
    cos21=np.sqrt(sum(myx*myx))
    cos22=np.sqrt(sum(myy*myy))
    return cos1/float(cos21*cos22)
```

```
if __name__ == '__main__':
```

```
    print
    print 'loading ...'
    print 'working',
    f1 = open(sampfn)
```

```
    try:
```

```
        f1_text = f1.read( )
```

```
        f1_text=unicode(f1_text,'utf-8')
```

```
    finally:
```



```

f1.close()
f1_seg_list = jieba.cut(f1_text)
# 第一个待测试数据

ftest1 = open(ftest1fn)
try:
    ftest1_text = ftest1.read()
    ftest1_text=unicode(ftest1_text,'utf-8')
finally:
    ftest1.close()
ftest1_seg_list = jieba.cut(ftest1_text)
# 第二个待测试数据
ftest2 = open(ftest2fn)
try:
    ftest2_text = ftest2.read()
    ftest2_text=unicode(ftest2_text,'utf-8')
finally:
    ftest2.close()
ftest2_seg_list = jieba.cut(ftest2_text)

# 读取样本文本
# 去除停用词，同时构造样本词的字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

test_words={}
all_words={}
for myword in f1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        test_words.setdefault(myword,0)
        all_words.setdefault(myword,0)
        all_words[myword]+=1

# 读取待测试文本
mytest1_words=copy.deepcopy(test_words)
for myword in ftest1_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest1_words.has_key(myword):
            mytest1_words[myword]+=1

mytest2_words=copy.deepcopy(test_words)
for myword in ftest2_seg_list:
    print ".",
    if not(myword.strip() in f_stop_seg_list):
        if mytest2_words.has_key(myword):
            mytest2_words[myword]+=1

```



```

# 计算样本与待测试文本的余弦相似度
sampdata=[]
test1data=[]
test2data=[]
for key in all_words.keys():
    sampdata.append(all_words[key])
    test1data.append(mytest1_words[key])
    test2data.append(mytest2_words[key])
test1simi=get_cossimi(sampdata,test1data)
test2simi=get_cossimi(sampdata,test2data)
print u"%s 与样本 [%s] 的余弦相似度 :%f"%(ftest1fn,sampfn,test1simi)
print u"%s 与样本 [%s] 的余弦相似度 :%f"%(ftest2fn,sampfn,test2simi)

```

## 12.3 朴素贝叶斯分类

贝叶斯是一种基于概率的学习算法，其性能可与决策树、神经网络等算法相媲美，是文本分类挖掘技术的典型代表。它以贝叶斯定理为基础，预测成员关系的可能性，由于其具有坚实的数学理论基础，并且能综合先验信息和数据样本信息，因此成为当前机器学习和数据挖掘的研究热点之一。朴素贝叶斯分类器是目前公认的一种简单有效的概率分类方法，这种分类方法具有非常高的计算效率，在某些应用问题上表现出较好的分类精度，因而被广泛地应用于文本挖掘领域。

### 12.3.1 算法描述

标准的朴素贝叶斯分类算法的执行过程如下：

- 1) 获取样本文本，将样本人工分类整理，并进行标记。
- 2) 对每个类别的样本文本进行中文分词。
- 3) 去除样本文本中垃圾词条。
- 4) 将整理后词条合成样本文本的特征组，分析并计算词条频率信息。例如：假设共有3个类别的文本，词条*i*在类别A、B、C中出现的次数分别为 $COUNT_i(A)$ 、 $COUNT_i(B)$ 、 $COUNT_i(C)$ ，每个类别的词条总数为 $WORDCOUNT(A)$ 、 $WORDCOUNT(B)$ 、 $WORDCOUNT(C)$ ，那么根据词每个类别的词条总数与词条在每个类别出现的次数就能计算词条频率，计算方式是：词语在每个类别出现的次数除以该类别的总词语数。

比如，某类别的词条总数是100个，而词条“冬天”出现了5次，那么“冬天”一词在该文件中的词频就是0.05（5/100）。

5) 根据词条频率信息，计算词条在各类别文本的先验概率。词条*i*的各类别先验概率计算公式为：

$$P_i(A)=COUNT_i(A)/WORDCOUNT(A)$$

$$P_i(B)=COUNT_i(B)/WORDCOUNT(B)$$

$$P_i(C)=COUNT_i(C)/WORDCOUNT(C)$$

- 6) 读取未知样本，进行中文分词，并去除垃圾词，然后形成样本特征组。

7) 将未知样本特征词条的先验概率代入朴素贝叶斯公式计算后验概率, 计算得到的最大概率的所属类别即为文本所属类别。

### 12.3.2 先验概率计算

同一条词条在不同类型的文本中出现的频率通常是不一样的, 很多词条只会在某些类别的文本中出现, 比如说“微软”、“谷歌”、“医保”、“乔布斯”等词条极少出现在战争类题材的文本中, 而“黑莓”、“3G”、“手机”、“电信”等词极少出现在健康类题材的文本中。词条先验概率计算通过提取不同类别中样本的词条, 分析其在所有类别中出现的概率, 它会以词条为键值, 生成词条先验概率哈希数组 (在 Python 中称为字典结构), 以供后期分类算法使用。

根据朴素贝叶斯的先验概率计算公式来看, 在后期计算中, 需要计算词条先验概率累乘。如果某词在某类型的样本中从来没有出现, 其概率为 0, 这样将会使累乘结果变为 0, 算法变得毫无意义。在计算先验概率后, 在每个词条的先验概率基础上加上一个适当的较小的概率值, 防止累乘出现 0。此外, 某词在某类型的所有样本中未出现, 不代表该词不会出现在该类型的所有文本中, 因此, 这个较小概率值非常有必要加入先验概率的计算中。

### 12.3.3 最大后验概率

对未知文本分类时, 需要计算后验概率, 对于在未知文本中出现的词条, 提取其在先验概率哈希数组中的概率值。然后分别计算不同类型哈希数组中出现的词条的先验概率累乘, 从而得到未知文本属于不同类型的后验概率, 其中, 最大概率所属类别即为未知文本所属类别。

### 12.3.4 算法实现

这里分别提取数量几乎相同的新闻文本作为样本, 这些样本属于汽车、财经、健康、教育、军事类新闻, 对样本进行分析。为了验证效果, 最后使用未在样本中出现的新闻正文链接进行测试, 分析该链接指向的新闻所属类别。

#### 1. 新闻爬取

提取新闻文本的原理与搜索引擎相同。首先, 通过类似爬虫的程序对新闻进行爬取, 分析新闻主页的新闻正文链接; 接着爬取新闻正文网页, 清理 HTML 标记; 然后形成文本样本, 为提高效率, 仅在内存中形成文本样本, 不在本地硬盘保存; 最后, 以内存数据为基础, 进行下一步分析。相关代码如下:

```
# 读取网上新闻搜索目录
txt_class=[]
myclassfl = open('ClassList.txt')
try:
    myclass_str = myclassfl.read()
    myclass_str=unicode(myclass_str,'gbk')
    myclass_text=myclass_str.split()
    for ii in xrange(0,len(myclass_text),2):
        print "...",
```

```

        txt_class.append((myclass_text[ii],myclass_text[ii+1]))
finally:
    myclassfl.close()

links=[]
# 分类别爬取网页,生成词条数据
for ci in xrange(0,len(txt_class)):
    print u"\n爬取%s类网页:%s" % (txt_class[ci][0],txt_class[ci][1])
    links.append([])
    pattern = re.compile(r'(.*)/\d+\.shtml')
    purl=txt_class[ci][1]
    page=urllib2.urlopen(purl)
    soup = BeautifulSoup(page)
    for link in soup.find_all('a'):
        mylink=link.get('href')
        match = pattern.match(mylink)
        if match and mylink.find("hd")<0:
            basestr="http://www.chinanews.com"
            if mylink.find("chinanews.com")<0:
                mylink=basestr+mylink
                print mylink
                links[ci].append(mylink)

```

## 2. 先验概率计算

提取不同类别中样本的词条,分析其在所有类别中出现的概率,生成以词条为键值的先验概率字典变量。此外,根据朴素贝叶斯的先验概率计算公式,在后期计算中,需要计算词条先验概率累乘,所以在每个词条的先验概率基础上加上一个适当的较小的概率值,防止累乘出现0。代码如下:

```

# 词条在每个样本中出现的次数
basegl=1e-8
wordybcount={}
lbcoun=np.zeros(len(yb_txt))
# 整理计算词条出现次数
for i in xrange(0,len(yb_txt)):
    for j in xrange(0,len(yb_txt[i])):
        for k in xrange(0,len(yb_txt[i][j])):
            my_word=yb_txt[i][j][k].encode('gbk')
            wordybcount.setdefault(my_word,np.repeat(0,len(yb_txt)).tolist())
            wordybcount[my_word][i]+=1
            lbcoun[i]+=1

# 计算词条先验概率
print u"\n计算词条概率"
ybg1={}

for my_word in wordybcount.keys():
    ybg1.setdefault(my_word,np.repeat(0.,len(yb_txt)).tolist())
    for ybii in xrange(0,len(yb_txt)):
        ybg1[my_word][ybii]=basegl+wordybcount[my_word][ybii]/float(lbcoun[ybii])
    print '.',

```

### 3. 后验概率计算

读取待分类文本的先验概率字典变量，提取每个词条的先验概率值，然后计算不同类型中出现的词条的先验概率累乘，最后得到待分类文本属于不同类型的后验概率。代码如下：

```
# 计算待分类文本后验概率
print u" 计算待分类文本后验概率 "
testgl=None
wordgl=None
testgl=np.repeat(1.,len(yb_txt))
for myword in ftest_seg_list:
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        myword=myword.encode('gbk')
        for i in xrange(0,len(yb_txt)):
            wordgl=ybgl.get(myword)
            if wordgl:
                if wordgl[i]<>0:
                    testgl[i]*=wordgl[i]
                    if np.min(testgl)<1e-50:
                        testgl*=1e20
                    if np.max(testgl)>1e100:
                        testgl/=float(1e30)
```

运行程序，读取网页 <http://www.chinanews.com/edu/2013/09-17/5296319.shtml> 和 <http://finance.chinanews.com/auto/2013/09-16/5290491.shtml>，以这两个网页为测试对象进行分类。执行结果如下：

```
.....
.....
读取待分类文本
http://www.chinanews.com/edu/2013/09-17/5296319.shtml 读取成功 .
计算待分类文本后验概率
http://www.chinanews.com/edu/2013/09-17/5296319.shtml
: 教育
计算待分类文本后验概率
http://finance.chinanews.com/auto/2013/09-16/5290491.shtml
: 汽车
```

从以上执行结果看，两个新闻链接被成功地划分到教育类新闻和汽车类新闻，分类效果良好。

本例中，样本数量较小，在实际应用中，每个类别应准备更多的样本文本。通常来说，一个分类效果较好的朴素贝叶斯算法，其每个类别的样本数量大致有 2000 ~ 10 000 个。近年来，基于朴素贝叶斯分类的改进算法越来越多，最普遍的是在算法中加入权重的影响，针对某些关键词或中心词加上权重，这种方法被称为加权朴素贝叶斯算法。

关于加权朴素贝叶斯算法的更多细节，可以查看在《厦门大学学报：自然科学版》2012 年第 4 期上刊登的饶丽丽等的文章《基于特征相关的改进加权朴素贝叶斯分类算法》，也可以在 Google 学术搜索 (<http://scholar.google.com.hk/schhp?hl=zh-CN>) 中以“加权朴素贝叶斯分类算法”为关键字进行搜索。



以下是完整的 Python 代码:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-
#code:myhaspl@qq.com
#12-5.py
#bayes 文本分类
# 本程序仅做机器学习研究
# 本程序对新闻爬取的工作原理与搜索引擎相同, 通过分析链接
# 直接搜索新闻, 计算词条概率

import numpy as np
import jieba
import urllib2
from bs4 import BeautifulSoup
import re

# 读取网上新闻搜索目录
txt_class=[]
myclassfl = open('ClassList.txt')
try:
    myclass_str = myclassfl.read()
    myclass_str=unicode(myclass_str, 'gbk')
    myclass_text=myclass_str.split()
    for ii in xrange(0,len(myclass_text),2):
        print ".",
        txt_class.append((myclass_text[ii],myclass_text[ii+1]))
finally:
    myclassfl.close()

links=[]
# 分类别爬取网页, 生成词条数据
for ci in xrange(0,len(txt_class)):
    print u"\n爬取 %s 类网页: %s" % (txt_class[ci][0],txt_class[ci][1])
    links.append([])
    pattern = re.compile(r'(.*)/\d+\.shtml')
    purl=txt_class[ci][1]
    page=urllib2.urlopen(purl)
    soup = BeautifulSoup(page)
    for link in soup.find_all('a'):
        mylink=link.get('href')
        match = pattern.match(mylink)
        if match and mylink.find("hd")<0:
            basestr="http://www.chinanews.com"
            if mylink.find("chinanews.com")<0:
                mylink=basestr+mylink
            print mylink
            links[ci].append(mylink)

# 提取正文内容
ybtxt=[]
print u"\n提取正文内容"
for ci in xrange(0,len(txt_class)):
```



```

ybtxt.append([])
print ".",
for mypage in links[ci]:
    try:
        my_page=urllib2.urlopen(mypage)
    except:
        continue
    my_soup = BeautifulSoup(my_page,from_encoding="gb2312")
    my_tt=my_soup.get_text("|", strip=True)
    my_txt=my_tt
    my_fs=u'正文|'
    my_fe1=u'【编辑|'
    my_fe2=u'标签|'
    zw_start=my_txt.find(my_fs)+8
    last_txt=my_txt[zw_start:len(my_txt)]
    zw_end=last_txt.find(my_fe1)
    if zw_end<0:
        zw_end=last_txt.find(my_fe2)
    page_content=my_txt[zw_start:zw_end]
    page_content=page_content.replace(r'_acK({aid:1807,format:0,mode:1,gid:1,serverbaseurl:"me.afp.chinanews.com/"});','').replace('|','').replace(r'_{aid:1805,format:0,mode:1,gid:1,serverbaseurl:"me.afp.chinanews.com/"};','').replace('cK();','')
    page_content=page_content.replace(u'1807:新闻通发页 大画','').replace(u'标签:','').replace(u'评论','').replace(u'正文 start 编辑姓名 start 编辑姓名','').replace(u'正文 start','')
    if len(page_content.strip())>0:
        try:
            print my_soup.title.string.encode('gb2312')
            page_content=my_soup.title.string+page_content
        except:
            print "...."
        finally:
            print "-done."
            ybtxt[ci].append(page_content)

# 分析正文内容
print u"\n分析正文内容 ..."

# 停用词字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

# 分类提取正文词条
print u"\n提取正文词条 ..."
yb_txt=[]
for ci in xrange(0,len(ybtxt)):
    yb_txt.append([])
    for cj in xrange(0,len(ybtxt[ci])):
        yb_txt[ci].append([])

```

```

my_str = ybtxt[ci][cj]
my_txt=jieba.cut(my_str)
for myword in my_txt:
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        yb_txt[ci][cj].append(myword)
print " ",

# 词条在每个样本中出现的次数
basegl=1e-10
wordybcnt={ }
lbcnt=np.zeros(len(yb_txt))
# 整理计算词条出现次数
for i in xrange(0,len(yb_txt)):
    for j in xrange(0,len(yb_txt[i])):
        for k in xrange(0,len(yb_txt[i][j])):
            my_word=yb_txt[i][j][k].encode('gbk')
            wordybcnt.setdefault(my_word,np.repeat(0,len(yb_txt)).tolist())
            wordybcnt[my_word][i]+=1
            lbcnt[i]+=1

# 计算词条先验概率
print u"\n 计算词条概率 "
ybg1={}

for my_word in wordybcnt.keys():
    ybg1.setdefault(my_word,np.repeat(0.,len(yb_txt)).tolist())
    for ybii in xrange(0,len(yb_txt)):
        ybg1[my_word][ybii]=basegl+wordybcnt[my_word][ybii]/float(lbcnt[ybii])
    print ' ',

# 读取待分类文本
print u"\n 读取待分类文本 "
fctestlinks=[]
fctestlinks.append(r'http://www.chinanews.com/edu/2013/09-17/5296319.shtml')
fctestlinks.append(r'http://finance.chinanews.com/auto/2013/09-16/5290491.shtml')
for mypage in fctestlinks:
    my_page=urllib2.urlopen(mypage)
    my_soup = BeautifulSoup(my_page,from_encoding="gb2312")
    my_tt=my_soup.get_text("|", strip=True)
    my_txt=my_tt
    my_fs=u' 正文 | '
    my_fe1=u' 【编辑】 '
    my_fe2=u' 标签 '
    zw_start=my_txt.find(my_fs)+8
    last_txt=my_txt[zw_start:len(my_txt)]
    zw_end=last_txt.find(my_fe1)
    if zw_end<0:
        zw_end=last_txt.find(my_fe2)
    page_content=my_txt[zw_start:zw_start+zw_end]
    page_content=page_content.replace(r'_acK({aid:1807,format:0,mode:1,gid:1,serverbaseurl:"me.afp.chinanews.com/"});','').replace('|','').replace(r'({aid:1805,format:0,mode:1,gid:1,serverbaseurl:"me.afp.chinanews.com/"}','').replace('cK();','')
    page_content=page_content.replace(u'1807: 新闻通发页 大画 ','').replace(u' 标签: ','').

```

```

replace(u'评论','').replace(u'正文 start 编辑姓名 start 编辑姓名','').replace(u'正文
start','')
page_content=my_soup.title.string+page_content
print u"%s 读取成功。"%mypage

# 计算待分类文本后验概率
print u" 计算待分类文本后验概率 "
testgl=None
wordgl=None
testgl=np.repeat(1.,len(yb_txt))
if len(page_content.strip())>0:
    ftest_seg_list = jieba.cut(page_content)
    for myword in ftest_seg_list:
        myword=myword.encode('gbk')
        if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>2:
            for i in xrange(0,len(yb_txt)):
                wordgl=ybgl.get(myword)
                if wordgl:
                    if wordgl[i]<>0:
                        testgl[i]*=wordgl[i]
                        if np.min(testgl)<1e-100:
                            testgl*=1e30
                        if np.max(testgl)>1e100:
                            testgl/=float(1e30)

# 计算最大归属概率
maxgl=0.
mychoice=0
for ti in xrange(0,len(yb_txt)):
    if testgl[ti]>maxgl:
        maxgl=testgl[ti]
        mychoice=ti
print "\n\n%s\n:%s"%(mypage,txt_class[mychoice][0])

```

## 12.4 自然语言处理

自然语言处理（NLP）是计算机科学领域与人工智能领域中的一个重要方向，是一门融语言学、计算机科学、数学于一体的学科，是计算机科学、人工智能、语言学研究计算机和人类（自然）语言之间的相互作用的工具。达到人类水平的自然语言处理，是一个人工智能的完全问题，NLP 研究能实现人与计算机之间用自然语言进行有效通信的各种理论和方法，相当于解决中央的人工智能问题，使计算机成为和人一样聪明或强大的人工智能。因此 NLP 的未来也会密切结合人工智能的发展而发展。

### 12.4.1 NLTK 简介

NLTK 是一个用来完成 NLP 的 Python 处理包。NLTK 致力于打造使用 Python 程序的人类语言工作平台，它提供了易于使用的接口、大量的英文语料库和词汇资源，连同一套文本

处理库的分类、标记、堵塞、标注、句法分析、语义推理、工业强度的 NLP 库包装及活跃的论坛，可用于 Windows、Mac OS X、Linux 等操作系统。

更为重要的是：NLTK 是一个免费的、开源的、社区驱动的项目。

此外，NLTK 虽然支持 unicode 编码的文本（中文可采用 unicode 编码，通常为 utf-8），但它是为处理英文文本而生的，因此 NLTK 对英文的支持比中文更好。鉴于上述原因，在实践中，可结合 jieba 中文分词组件来完成 NLP 的相关工作。

## 12.4.2 NLTK 与 jieba 的配置

### 1. NLTK 的安装与配置

首先打开 NLTK 的官网 <http://www.nltk.org/>，下载相关平台的安装包并安装好，然后在 Python 交互解释器下执行下面的语句：

```
>>> import nltk
>>> nltk.download()
```

最后，将出现 NLTK 包的下载界面，选择下载目录后，按 Download 键（如图 12-4 所示）。下载完毕后，在 Python 的安装目录下新建 nltk\_data 文件夹，将下载的文件拷入其中。以 WinPython 为例，笔者的 WinPython 安装目录如下：

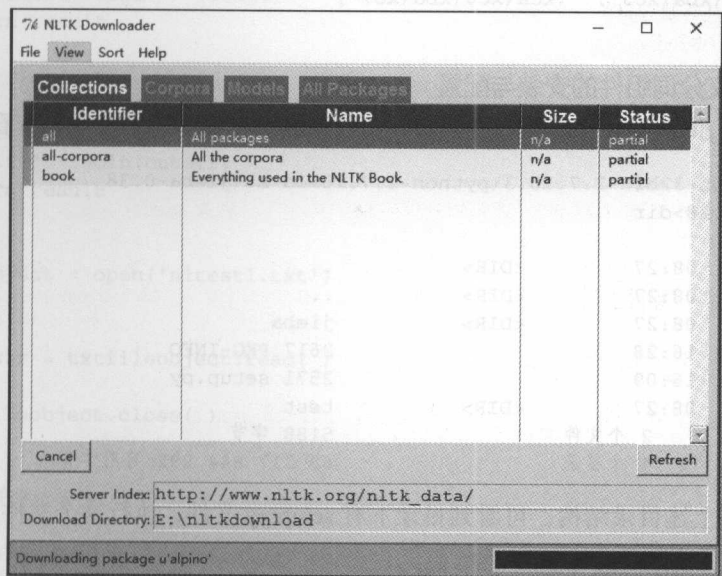


图 12-4 NLTK 包下载

E:\WinPython-32bit-2.7.10.3\python-2.7.10\nltk\_data

将 nltk\_data 拷入其中后，效果如图 12-5 所示。

上述步骤全部完成后，可在 Python 交互解释器下输入如下代码，进行测试：

此电脑 > data1 (E:) > WinPython-32bit-2.7.10.3 > python-2.7.10 > nltk_data			
名称	修改日期	类型	
chunkers	2016/1/26 17:28	文件夹	
corpora	2016/1/26 17:31	文件夹	
grammars	2016/1/26 17:31	文件夹	
help	2016/1/26 17:31	文件夹	
models	2016/1/26 17:31	文件夹	
stemmers	2016/1/26 17:31	文件夹	
taggers	2016/1/26 17:31	文件夹	
tokenizers	2016/1/26 17:31	文件夹	

图 12-5 nltk\_data 拷贝

```
>>> import nltk
>>> sentence = ""At eight o'clock on Thursday morning
... .. Arthur didn't feel very good.""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', 'o'clock', 'on', 'Thursday', 'morning', '...', 'Arthur', 'did',
'n't', 'feel', 'very', 'good', '.']
>>>
以空格分隔的中文
>>> mystr="您好 世界"
>>> tokens = nltk.word_tokenize(mystr)
>>> tokens
['\xc4\xfa\xba\xc3', '\xca\xc0\xbd\xe7']
>>>
```

## 2. jieba 中文分词组件的安装与配置

首先，下载 jieba 组件并解压缩。接着，打开控制台，输入以下命令来查看目录的结构：

```
E:\WinPython-32bit-2.7.10.3\python-2.7.10>cd E:\jieba-0.38
E:\jieba-0.38>dir
...
2016/01/27  08:27          <DIR>          .
2016/01/27  08:27          <DIR>          ..
2016/01/27  08:27          <DIR>          jieba
2015/12/16  16:28          2617 PKG-INFO
2015/12/16  16:09          2571 setup.py
2016/01/27  08:27          <DIR>          test
                        2 个文件          5188 字节
                        4 个目录          69 317 844 992 可用字节
```

然后，观察上述目录结构，可看到目录下有 setup.py 文件，执行这个文件进行安装。

```
E:\jieba-0.38>python setup.py install
...
...
byte-compiling E:\WinPython-32bit-2.7.10.3\python-2.7.10\Lib\site-packages\
jieba\__main__.py to __main__.pyc
running install_egg_info
Writing E:\WinPython-32bit-2.7.10.3\python-2.7.10\Lib\site-packages\jieba-0.38-
py2.7.egg-info
```



### 12.4.3 中文分词并标注词性

NLTK 和 jieba 安装配置完毕后,可测试一个中文分词和词性标注的程序(如 12-6.py 所示)。

```
#encoding=utf-8
#--coding:utf-8--
#code by myhaspl
# 分词, 词性
#12-6.py
from __future__ import unicode_literals

import nltk
import sys
sys.path.append("../")

import jieba
from jieba import posseg

def cutstrpos(txt):
    # 分词 + 词性
    cutstr = posseg.cut(txt)
    result=""
    for word, flag in cutstr:
        result+=word+"/"+flag+' '
    return result

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

# 读取文件
txtfileobject = open('nltest1.txt')
textstr=""
try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()

# 中文分词并标注词性
posstr=cutstrpos(filestr)
strtag=[nltk.tag.str2tuple(word) for word in posstr.split()]
for word,tag in strtag:
    print word,"/",tag,"|",

# 进入语料库
cutstr=cutstring(filestr)
mytext=nltk.text.Text(cutstr)
# 在该语料库中查找包括 "人" 的语句
print(mytext.concordance(u"人"))
```

观察程序 12-6.py 可发现，它演示了以下两种分词方式。

### (1) 纯中文分词

`cutstring` 函数负责文本的中文分词，通过直接调用 `jieba` 模块的 `cut` 函数来完成，分割的词使用空格分离的方式进行标注，代码片断如下所示：

```
def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result
```

### (2) 中文分词并标注词性

`cutstrpos` 函数负责文本的中文分词，并标注词性，通过直接调用 `posseg` 模块的 `cut` 函数来完成，该 `cut` 函数将返回一个列表，列表中是由形如“(词，词性)”格式的元组组成，因此，分割的词使用空格分离的方式进行标注的同时，还需要使用“/”分割符（也可使用其他非空格的分割字符）将词与词性进行分离。`cutstrpos` 函数代码片断如下所示：

```
def cutstrpos(txt):
    # 分词 + 词性
    cutstr = posseg.cut(txt)
    result=""
    for word, flag in cutstr:
        result+=word+"/"+flag+' '
    return result
```

此外，`Concordance` 函数可显示指定的词在某语料库中的出现情况，并显示一些上下文。

程序 12-6.py 的运行效果如下：

```
据 / P | 国外 / S | 媒体报道 / N | , / X | 美国 / NS | 科学家 / N | 近日 / T | 获得 /
V | 了 / UL | 2800 / M | 万美元 / M | ( / X | 约合 / VN | 1.84 / M | 亿 / M | 人民币 /
N | ) / X | 的 / UJ | 研究 / VN | 经费 / VN | , ..... / X | 我们 / R | 迟早会 / NR |
设计 / VN | 出 / V | 一款 / M | 能够 / V | 媲美 / V | 、 / X | 甚至 / D | 超越 / V | 人类
/ N | 的 / UJ | 计算机系统 / N | 。 / X | Displaying 11 of 11 matches:
0 0 万 美 元 ( 约 合 1 . 8 4 亿 人 民 币 ) 的 研 究 经 费 ,
用 于 ..... 它 的 规 模 类 似 于 人 类 基 因 组 计 划 。 该 项 目 的
领 出 一 款 能 够 媲 美 、 甚 至 超 越 人 类 的 计 算 机 系 统 。
```

观察上述运行效果，第一段是中文分词与词性标注的结果，第二段是将文本加入 NLTK 语料库，并在语料库中查找含有“人”的语句的结果。



**提示** NLTK 的语料库对英文的支持非常好，但对中文的支持有限，因此，如果使用 NLTK 语料库对中文进行处理，请慎重使用，反复调试。

## 12.4.4 词特征指标分析

### 1. 词频统计

程序 12-7.py 演示了如何调用 NLTK 模块的函数进行词频统计。

```

#--coding:utf-8--
#code by myhaspl
# 词频分析
#12-7.py
from __future__ import unicode_literals
from __future__ import division

import nltk

import sys
sys.path.append("../")

import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

# 读取文件
txtfileobject = open('nltest1.txt','r')

try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()

cutstr=cutstring(filestr)
tokenstr=nltk.word_tokenize(cutstr)
# 全文总词数
print u" 词总数:",
print len(tokenstr)
# 共出现多少词
print u" 共出现词数:",
print len(set(tokenstr))
# 词汇条目排序表
print u" 词汇条目排序表 "
for word in sorted(set(tokenstr)):
    print word,
print
# 每个词的平均使用次数
print u" 每个词的平均使用次数:",
print len(tokenstr)/len(set(tokenstr))
# 统计词频
fdist1=nltk.FreqDist(tokenstr)
for key,val in sorted(fdist1.iteritems()):
    print key,val,
print
print u"..... 计算机系统出现的次数 ....."

```

```
print fdist1[u' 计算机系统 ']
```

# 统计出现最多的前 5 个词

```
print
print u"..... 统计出现最多的前 5 个词 ....."
fdist1=nltk.FreqDist(tokenstr)
for key,val in sorted(fdist1.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:5]:
    print key,val
```

观察程序 12-8.py, 该程序依次执行了以下操作:

### (1) 读取文件

通过调用 Python 的 open 函数来打开文件, read 函数来读取文件, 无论读取文件成功与否都关闭文件对象, 代码片断如下所示:

```
txtfileobject = open('nltest1.txt','r')
try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()
```

### (2) 中文分词

首先, 调用 cutstring 函数, 从而使用 jiaba 的 cut 函数完成中文分词, 其中, 每个词都用空格进行分割; 然后, 针对空格分割的分词结果, 调用 NLTK 模块的 word\_tokenize 函数进行 NLTK 方式的二次分词, 最终生成 NLTK 模块要求的中文分词格式, 即: 以中文词为元素, 组成词语列表。代码片断如下所示:

```
cutstr=cutstring(filestr)
tokenstr=nltk.word_tokenize(cutstr)
```



**提示** NLTK 模块的分词是针对英文分词而设计的, 而英文分词相对比较简单, 通常句子中的英语单词是用空格来分隔的, 因此, 使用 NLTK 完成中文分词, 需要模拟英文分词的方式来进行, 即: 先使用空格将中文词组分割形成字符串后, 再送往 NLTK 分词函数做进一步处理。

### (3) 统计词出现的次数

首先, 通过调用 len 函数, 对 NLTK 分词形成的词列表中的元素总数 (即词的总数量) 进行统计; 然后, 调用 set 函数将 NLTK 分词结果转换成集合, 统计集合中出现的词数。代码片断如下所示:

```
# 全文总词数
print u" 词总数 :",
print len(tokenstr)
# 共出现多少词
print u" 共出现词数 :",
print len(set(tokenstr))
```

#### (4) 统计词频

首先,调用 NLTK 模块的 FreqDist 函数,生成 NLTK 词频字典对象,该字典的键为词,值为该词出现的次数;然后,调用该词频对象的 iteritems() 返回迭代对象,遍历所有的词。代码片断如下所示:

```
# 统计词频
fdist1=nltk.FreqDist(tokenstr)
for key,val in sorted(fdist1.iteritems()):
    print key,val,print u"共出现词数:",
```

#### (5) 按词频大小分析中文词

首先,生成 NLTK 词频字典;然后,依据字典的值(即词频)对字典的键(即中文词语)进行从大到小的排序。下面的代码片断演示的是分析出现最多的前 5 个词:

```
fdist1=nltk.FreqDist(tokenstr)
for key,val in sorted(fdist1.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:5]:
    print key,val
```

程序 12-7.py 的运行效果如下:

```
词总数 : 386
共出现词数 : 216
词汇条目排序表
1.84 2800 CBS Cox David IARPA SEAS . 、 。 一个 一半 一款 一点 一项 万美元 上 不然 与
东西 中 中心 为了 为何 之后 之间 也 了 .....
每个词的平均使用次数为 1.78703703704
1.84 1 2800 1 CBS 1 Cox 1 David 1 IARPA 1 SEAS 2 . 1 、 7 。 13 一个 2 一半 1 一款
2 一点 1 一项 1 万美元 1 上 1 不然 1 与 5 东西 1 中 1 中心 1 为了 1 为何 1 之后 1 之间 2 也
1 了 6 于 1 人员 1 .....
..... 计算机系统出现的次数 .....
3
..... 统计出现最多的前 5 个词 .....
的 25
, 23
。 13
人类 8
、 7
```

观察上述 12-7.py 的运行结果,程序依次输出了词总数、共出现词数、词汇条目排序表、每个词的平均使用次数、“计算机系统”这个词出现的次数、出现最多的前 5 个词。

## 2. 词频与长词分析

程序 12-8.py 演示了如何调用 NLTK 模块的函数进行词频统计:

```
##--coding:utf-8--
#code by myhaspl
#12-8.py
from __future__ import unicode_literals
from __future__ import division
```



```

import nltk

import sys
sys.path.append("../")

import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

# 读取文件
txtfileobject = open('nlttest2.txt','r')
try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()

cutstr=cutstring(filestr)
tokenstr=nltk.word_tokenize(cutstr)

fdist1=nltk.FreqDist(tokenstr)
# 只出现了1次的低频词
print "---- 只出现了1次的低频词 ----"
for word in fdist1.hapaxes():
    print word,
# 找出文本中的长词
print
print "---- 文本中的长词 ----"
for word in [w for w in set(tokenstr) if len(w)>3]:
    print word,
# 找出文本中出现了2次以上的长词
print
print "---- 文本中出现了2次以上的长词 ----"
for word in [w for w in set(tokenstr) if len(w)>3 and fdist1[w]>2]:
    print word,

```

程序 12-8.py 中有以下几个关键知识点。

#### (1) 低频词

NLTK 将只出现过 1 次的词作为低频词，可通过调用 hapaxes 函数分析低频词。如下面的代码片断所示：

```

# 只出现了1次的低频词
print "---- 只出现了1次的低频词 ----"
for word in fdist1.hapaxes():
    print word,

```

## (2) 长词

可将长度超过 3 的中文词视为长词，如下面的代码片断所示，首先通过 `len` 函数找到文本中的长词，然后结合词频字典的值来寻找出现了 2 次以上的长词。

```
# 找出文本中的长词
print
print "---- 文本中的长词 ----"
for word in [w for w in set(tokenstr) if len(w)>3]:
    print word,
# 找出文本中出现了 2 次以上的长词
print
print "---- 文本中出现了 2 次以上的长词 ----"
for word in [w for w in set(tokenstr) if len(w)>3 and fdist1[w]>2]:
    print word,
```

程序 12-8.py 的输出结果如下：

```
---- 只出现了 1 次的低频词 ----
无意识 加快 一方面 特性 电视观众 窗 圣哲 神经科学 尽可能 团队 置于 繁重 经 干预 显然 下丘脑
关系 中带 正确 .....
---- 文本中的长词 ----
大脑皮层 电视观众 神经科学 另一方面 与此同时 .....
---- 文本中出现了 2 次以上的长词 ----
持续时间
```

## 3. 搭配词分析

程序 12-9.py 演示了如何调用 NLTK 模块的函数进行搭配词分析：

```
#!/usr/bin/env python
#--coding:utf-8--
#code by myhaspl
#12-9.py
from __future__ import unicode_literals
from __future__ import division

import nltk

import sys
sys.path.append("../")

import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

# 读取文件
```

```

txtfileobject = open('nltest2.txt','r')

try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()

import re
cutstr=cutstring(filestr)
tokenstr=nlk.word_tokenize(cutstr)
fdist1=nlk.FreqDist(tokenstr)

bigramcolloc=nlk.collocations.BigramCollocationFinder.from_words(tokenstr)
print "---- 出现最频繁的前 10 个词 ----"
fdist1=bigramcolloc.word_fd
for key,val in sorted(fdist1.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:10]:
    print key,":",val

print "---- 只出现了 1 次的低频词 ----"
fdist1=bigramcolloc.word_fd
for w in fdist1.hapaxes():
    print w.encode("utf-8"),"|"

# 找出文本中的搭配词
print
print "---- 找出双连搭配词 ----"
bigramwords=nlk.bigrams(tokenstr)
for fw,sw in set(bigramwords):
    print fw," ",sw,"|",

print
print "---- 双连搭配词及词频 ----"
for w,c in sorted(bigramcolloc.ngram_fd.iteritems(),key=lambda
x:(x[1],x[0]),reverse=True):
    fw,sw=w
    print fw," ",sw,">=",c,"|",
print

trigramcolloc=nlk.collocations.TrigramCollocationFinder.from_words(tokenstr)
print "---- 三连搭配词 ----"
for fw,sw,tw in trigramcolloc.ngram_fd:
    print fw.encode("utf-8")," ",sw.encode("utf-8")," ",tw.encode("utf-8"),"|"
print

```

程序 12-9.py 中有以下几个关键知识点。

#### (1) 双连搭配词

将两个经常在一起使用的词语合并为一个词组，并称为双连搭配词，比如：“通过”、“扫描”这两个词经常在一起使用，可合并为“通过扫描”的词组。可通过调用 `bigrams` 函数来寻找所有的双连搭配词。代码片断如下所示：

```
bigramwords=nlk.bigrams(tokenstr)
```

```
for fw,sw in set(bigramwords):
    print fw," ",sw,"|",
```

## (2) 三连搭配词

将三个经常在一起使用的词语合并为一个词组，并称为三连搭配词，比如：“发现”、“任何”、“异样”这三个词经常在一起使用，可合并为“发现任何异样”的词组。可通过调用模块 `nltk.collocations.TrigramCollocationFinder` 的 `from_words` 函数来寻找所有的三连搭配词，并统计词频。代码片断如下所示：

```
trigramcolloc=nltk.collocations.TrigramCollocationFinder.from_words(tokenstr)
for fw,sw,tw in trigramcolloc.ngram_fd:
    print fw.encode("utf-8")," ",sw.encode("utf-8")," ",tw.encode("utf-8"),"|",
    print
```

## (3) 词频

可通过调用模块 `nltk.collocations.BigramCollocationFinder` 的 `from_words` 函数来寻找所有的双连搭配词，并统计词频。

其中，可通过 `word_fd` 方法返回单个词的词频，代码片断如下所示：

```
bigramcolloc=nltk.collocations.BigramCollocationFinder.from_words(tokenstr)
print "---- 出现最频繁的前 10 个词 ----"
fdist1=bigramcolloc.word_fd
for key,val in sorted(fdist1.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:10]:
    print key,":",val
```

也可以通过 `iteritems` 方法返回双连搭配词的词频，代码片断如下所示：

```
bigramcolloc=nltk.collocations.BigramCollocationFinder.from_words(tokenstr)
for w,c in sorted(bigramcolloc.ngram_fd.iteritems(),key=lambda x:(x[1],x[0]),
reverse=True):
    fw,sw=w
    print fw," ",sw,">","c","||",
    print
```

程序 12-9.py 的运行结果如下：


```
---- 出现最频繁的前 10 个词 ----
的 : 131
, : 99
。 : 64
在 : 30
大脑 : 28
...
---- 只出现了 1 次的低频词 ----
无意识 | 加快 | 一方面 | 特性 | 电视观众 | 窗 | 圣哲 | 神经科学 | 尽可能 | 团队 | 置于 |
繁重 | 经 | 干预 | 显然 | ..... 神经学家 | 前 |
---- 找出双连搭配词 ----
..... 声音 的 | 例如 在 | 振荡 活动 | 的 区别 | 就 像 | 控制 着 | 和
从来 | 如果 两件事 | 这 就 | 随机 组合 .....
```

```

---- 双连搭配词及词频 ----
..... 不同 速度 => 2 || 不 超过 => 2 || 下 一个 => 2 || 一项 试验 => 2 || 一
个 等级 => 2 .....
---- 三连搭配词 ----
..... 大脑 中 发生 | 生活 在 太 | 他们 观察 一个 | 不 清楚 这些 | 建
立 在 过去 | 维特曼 设计 出 .....

```

观察上述运行结果，程序 12-9.py 依次输出了出现最频繁的前 10 个词、只出现了 1 次的低频词、双连搭配词、双连搭配词及词频、三连搭配词。

 **提示** 12.4 节中有些代码并没有对标点符号做任何处理，比如 12-9.py 没有对标点进行过滤，读者可参考本章前面的部分，将标点符号作为停用词进行删除处理。但有一点必须要注意：标点符号并非完全没有意义，在 NLP 过程的初期并不一定要过滤标点符号，例如，标点符号可以作为词组、句子甚至口语文本分割的标志等。

#### 4. 词详细指标分析

程序 12-10.py 演示了如何调用 NLTK 模块的函数进行不同指标的词频分析、某词汇指标分析、样本特征分析、频率分析等。

```

#--coding:utf-8--
#code by myhaspl
#12-10.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

# 读取文件
txtfileobject = open('test2.txt','r')

try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()

cutstr=cutstring(filestr)

```



```

tokenstr=nlk.word_tokenize(cutstr)

fdist=nlk.FreqDist(tokenstr)

# 以词长为元素, 计算不同词长的频率
print "---- 词频 ----"
fdist1=nlk.FreqDist([len(w) for w in tokenstr])
for w,c in fdist1.items():
    print w,">","c","||",
# 词长
print
print "---- 词长 ----"
print fdist1.keys()

# 词
print
print u"--- 词频 ---"
fdist2=nlk.FreqDist(tokenstr)
for w,c in fdist2.items():
    print w,">","c","||",

print
print u"--- 无意识出现的次数 ---"
print fdist2[u"无意识"]
print u"--- 神经学家出现的次数 ---"
print fdist2[u"神经学家"]

# 其他基本指标
sample=cutstring(u"据悉, 这辆汽车绰号野兽, 野兽很可能于 2017 年 1 月份美国第 45 任总统就职时
使用。目前, 野兽的详细规格都属于绝密信息, 但谍照显示野兽采用了凯迪拉克的最新护栅和前灯设计。")
tokenstr=nlk.word_tokenize(sample)
fdist3=nlk.FreqDist(tokenstr)
print u"--- 美国出现的次数 ---"
print fdist3[u"美国"]
print u"--- 样本总数 ---"
print fdist3.N()
print u"--- 数值最大的样本 ---"
print fdist3.max()
# 频率分布表
fdist3.tabulate()
# 频率分布图
fdist3.plot()
# 前 10 个高频词的累积频率分布图
fdist3.plot(10,cumulative=True)

```

程序 12-10.py 中有以下几个关键知识点。

1) 以词长作为指标进行词频分析。通过 len 函数取出词长, 以词长为元素形成列表, 将列表作为 FreqDist 函数的参数, 返回以词长为指标的词频字典, 代码如下所示:

```
fdist1=nlk.FreqDist([len(w) for w in tokenstr])
```

```
for w,c in fdist1.items():
    print w,"=>",c,"||",
```

2) 频率分布。调用 `tabulate` 函数输出词频的分布情况, 调用 `plot` 函数绘制词频分布图和累积频率分布图。代码如下所示:

```
# 频率分布表
fdist3.tabulate()
# 频率分布图
fdist3.plot()
# 前 10 个高频词的累积频率分布图
fdist3.plot(10,cumulative=True)
```

3) `pylab` 绘图的中文乱码处理。通过指定 `pylab` 字体的方式可避免绘图出现中文乱码。代码如下所示:

```
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
```

程序 12-10.py 执行结果如下:

```
---- 词频 ----
1 => 750 || 2 => 864 || 3 => 80 || 4 => 28 || 5 => 2 || 6 => 1 ||
---- 词长 ----
[1, 2, 3, 4, 5, 6]
--- 词频 ---
要 => 2 || 大脑皮层 => 2 || 一切 => 3 || 无意识 => 1 || 加快 => 1 || 一方面 => 1 ||
通过 => 2 || 特性 => 1 || .....|| 情绪 => 1 || 是否 => 1 || 蜂鸣 => 1 || 他们 => 2 ||
近距离 => 1 || 起 => 1 || 神经学家 => 1 || 前 => 1 || 能够 => 7 ||
--- 无意识出现的次数 ---
1
--- 神经学家出现的次数 ---
1
--- 美国出现的次数 ---
1
--- 样本总数 ---
49
--- 数值最大的样本 ---
..... 可能 年 详细 时 任 凯迪拉克 都
..... 1 1 1 1 1 1 1
```

此外, 程序 12-10.py 绘制了如图 12-6 所示的词频率分布图。

观察图 12-6, 上面的曲线为累积频率分布曲线, 观察该曲线, “,”、“野兽”这两个词共使用了 8 次, “,”、“野兽、“。”这 3 个词共使用了 10 次。下面的曲线是频率分布曲线, 观察该曲线可以看出, “显示”一词出现了 1 次, “的”一词出现了 2 次, “野兽”一词出现了 4 次。



**提示** 累积频率 (Cumulative Percentage) 是指, 按某种标志对数据进行分组后, 分布在各组内的数据个数称为频数或次数, 各组频数与全部频数之和的比值称为频率或比重。为了统计分析的需要, 有时需要观察某一数值以下或某一数值以上的频率之和, 叫作

累积频率，或称为对频率的累计。从变量值小的一方向变量值大的一方累加，称为向上累积，反之为向下累积。频率的最终累积值为 100%。设  $x_1 < x_2 < \dots < x_m$  是不重复的样本值， $m < n$ ，把样本值小于等于某个样本数据  $x_i$  的频率累加起来，就可得到小于等于  $x_i$  的累积频率。

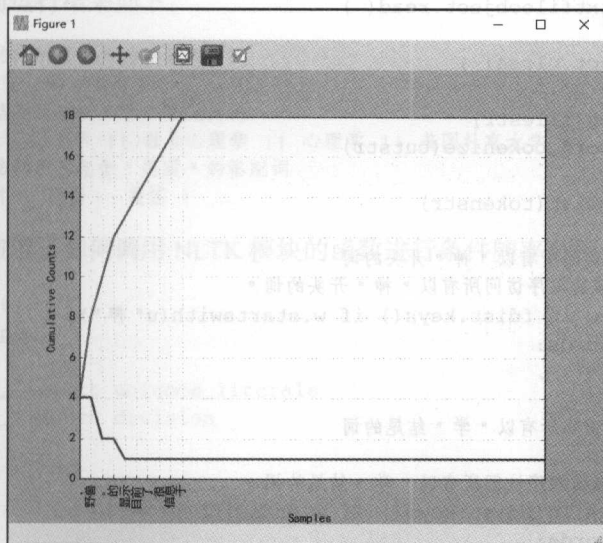


图 12-6 词频率分布图

程序 12-11.py 演示了如何调用 NLTK 模块的函数对词缀、包含词等情况进行分析：

```

#--coding:utf-8--
#code by myhaspl
#12-11.py

from __future__ import unicode_literals
from __future__ import division
import pylab

import nltk

import sys
sys.path.append("../")
pylab.rcParams['font.sans-serif']=['SimHei']
import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)

```

```

result=" ".join(cutstr)
return result

# 读取文件
txtfileobject = open('nltest2.txt','r')

try:
    filestr = txtfileobject.read()
finally:
    txtfileobject.close()

cutstr=cutstring(filestr)
tokenstr=nltk.word_tokenize(cutstr)

fdist=nltk.FreqDist(tokenstr)

# 以词频递减的顺序访问所有以 " 神 " 开头的词
print " 以词频递减的顺序访问所有以 " 神 " 开头的词 "
mywords=[w for w in fdist.keys() if w.startswith(u" 神 ")]
for word in mywords:
    print word,"||",

# 以词频递减的顺序访问所有以 " 学 " 结尾的词
print
print " 以词频递减的顺序访问所有以 " 学 " 结尾的词 "
mywords=[w for w in fdist.keys() if w.endswith(u" 学 ")]
for word in mywords:
    print word,"||",

# 以词频递减的顺序访问所有包含 " 美国 " 的搭配词
print
print " 以词频递减的顺序访问所有包含 " 美国 " 的搭配词 "
bigramwords=nltk.bigrams(tokenstr)
mywords=[w for w in set(bigramwords) if u" 美国 " in w]
for fw,sw in mywords:
    print fw," ",sw,"|",

```

程序 12-11.py 中有以下几个关键知识点。

1) 词缀。通过 `endswith` 函数对词后缀进行分析。代码如下所示：

```

print " 以词频递减的顺序访问所有以 " 学 " 结尾的词 "
mywords=[w for w in fdist.keys() if w.endswith(u" 学 ")]
for word in mywords:
    print word,"||",

```

通过 `startswith` 函数对词的前缀进行分析。代码如下所示：

```

print " 以词频递减的顺序访问所有以 " 神 " 开头的词 "
mywords=[w for w in fdist.keys() if w.startswith(u" 神 ")]
for word in mywords:
    print word,"||",

```

2) 包含词。通过 in 对包含词进行分析。代码如下所示:

```
bigramwords=nltk.bigrams(tokenstr)
mywords=[w for w in set(bigramwords) if u"美国" in w]
for fw,sw in mywords:
    print fw," ",sw,"|",
```

程序 12-11.py 的运行结果如下:

```
以词频递减的顺序访问所有以 "神" 开头的词
神经科学 || 神经节 || 神经学家 ||
以词频递减的顺序访问所有以 "学" 结尾的词
神经科学 || 大学 || 光学 || 社会心理学 || 心理学 || 美国杜克大学 ||
以词频递减的顺序访问所有包含 "美国" 的搭配词
美国 得克萨斯州 | 根据 美国 |
```

程序 12-12.py 演示了如何调用 NLTK 模块的函数进行条件频率分析:

```
#!/usr/bin/env python
#--coding:utf-8--
#code by myhaspl
#12-12.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

samples=[('nltest1.txt',u'科技'),('nltest2.txt',u'科技'),('nltest3.txt',u'财经'),('nltest4.txt',u'财经')]
samplewords=[]
for (filename,categories) in samples:
    # 读取文件
    txtfileobject = open(filename,'r')
    try:
        filestr = txtfileobject.read()
    finally:
        txtfileobject.close()

    cutstr=cutstring(filestr)
    tokenstr=nltk.word_tokenize(cutstr)
```



```

mywords=[w for w in tokenstr]
for word in mywords:
    samplewords.append((categories,word))
# 条件频率, 每个词汇在不同分类中出现的频率
print "-----"
cfd=nlk.ConditionalFreqDist(samplewords)
fdist=cfd[u' 财经 ']
for word in fdist:
    print word
print "----- 流动性出现次数 -----"
print cfd[u' 财经 '][u' 流动性 ']
print "----- 条件: 分类 -----"
for cnd in cfd.conditions():
    print cnd
print "-----"
# 频数最大的样本
print cfd[u' 财经 '].max()
# 条件频率分布
print "----- 条件频率分布表 -----"
cfd.tabulate(title=u' 条件频率分布表 ',conditions=[u' 科技 ',u' 财经 '])
cfd.plot(title=u' 条件频率分布图 ',conditions=[u' 科技 ',u' 财经 '])

```

程序 12-12.py 中有以下几个关键知识点。

1) 条件频率字典构造。可使用 NLTK 模块的 ConditionalFreqDist 函数构造条件频率字典, 条件频率字典与 FreqDist 函数构造的频率字典不同 (如 12-11.py 所示), 它在频率的基础上增加了文本类别, 这样就可以实现分类的频率分析了。

此外, ConditionalFreqDist 函数的参数是一个列表, 列表中的每个元素均为带类别标志的元组, 格式为: (类别, 词汇)。代码如下所示:

```

for word in mywords:
    samplewords.append((categories,word))

```

2) 条件频率字典访问。生成的条件频率字典属于二维结构, 第一维是类别, 第二维是词汇, 下面的代码演示了分析访问财经类文本中的“流动性”一词的频率。

```

print "----- 流动性出现的次数 -----"
print cfd[u' 财经 '][u' 流动性 ']

```

3) 条件频率分布。使用 tabulate 函数绘制条件频率分布表, 使用 plot 函数绘制条件频率分布图。

```

cfd.tabulate(title=u' 条件频率分布表 ',conditions=[u' 科技 ',u' 财经 '])
cfd.plot (title=u' 条件频率分布图 ',conditions=[u' 科技 ',u' 财经 '])

```

程序 12-12.py 首先输出财经类的所有词汇及财经类中“流动性”的数量; 然后输出该字典内包含的类别 (条件), 最后输出频数最大的样本及频率分布的情况。运行结果如下:

```

...
...
...
体量

```

----- 流动性出现的次数 -----

----- 条件：分类 -----

---

----- 条件频率分布表 -----

高级 高达 高速 鸡尾酒 鸣声 黑暗 鼓励 ( ), : ; ?

科技	.....	1	1	0	1	1	1	0	6	6	122	2	2	2
财经	.....	0	1	2	0	0	0	2	3	3	169	4	1	0

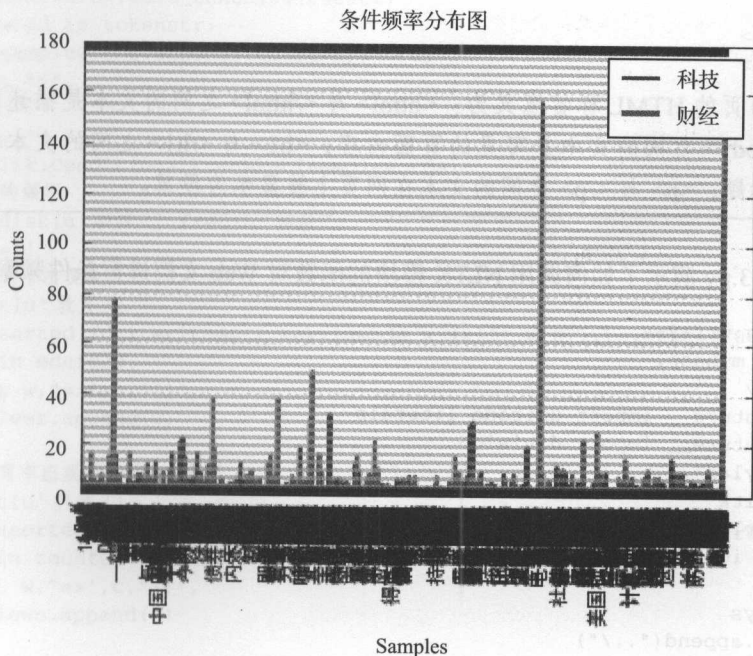


图 12-7 条件频率分布图

### 12.4.5 Web 文档分析

Web 文档即网页，它是构成网站的基本元素，是承载各种网站应用的平台，Web 文档是一个包含 HTML 标签的纯文本文件，它可以存放在世界某个角落的某一台计算机中，是万维网中的一“页”，其格式为超文本标记语言（标准通用标记语言的一个应用），通常 Web 文档的文件扩展名为 .html 或 .htm）。



HTML 是用来描述网页的一种语言，是超文本标记语言（Hyper Text Markup Language），但它不是一种编程语言，而是一种标记语言（markup language），标记语言是一套标记标签（markup tag）。

HTML 使用标记标签来描述网页，HTML 标记标签通常被称为 HTML 标签（HTML tag）。HTML 标签是由尖括号包围的关键词组成的，比如“<html>”。HTML 标签通常是成对出现的，比如“<b>”和“</b>”，标签对中的第一个标签是开始标签，第二个标签是结束标签，开始和结束标签也称为开放标签和闭合标签。比如下面这段网页：

```
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

上述网页的 HTML 标签意义为：<html> 与 </html> 之间的文本是描述网页；<body> 与 </body> 之间的文本是可见的页面内容；<h1> 与 </h1> 之间的文本在网页上被显示为标题；<p> 与 </p> 之间的文本在网页上被显示为段落。

程序 12-13.py 演示了如何调用 NLTK 模块的函数对 Web 文档进行条件频率分析：

```
#!/usr/bin/env python
#--coding:utf-8--
#code by myhaspl
#12-13.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib
from bs4 import BeautifulSoup

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba
```

```
def cutstring(txt):
```

```
    # 分词
```

```
    cutstr = jieba.cut(txt)
```

```
    result=" ".join(cutstr)
```

```
    return result
```

```
urls=[(u"科技", "http://tech.163.com/16/0203/06/BESLRP50000915BD.html"), (u"科技
```

```

", "http://tech.163.com/16/0202/01/BEPHEI120009405H.html"), (u"科技", "http://tech.163.com/16/0203/03/BESBB73B000915BD.html"), (u"科技", "http://tech.163.com/16/0203/03/BESAGOPB000915BD.html"), (u"教育", "http://edu.163.com/16/0203/05/BESI2S7500294NE9.html"), (u"教育", "http://kids.163.com/16/0118/06/BDJEMJ3H00294MO6.html"), (u"教育", "http://edu.163.com/16/0128/05/BED4NHBB00294NE9.html"), (u"教育", "http://edu.163.com/16/0202/01/BEPHFQ1800294IIH.html")]
samplewords=[]
print "|=",
for (category,myurl) in urls:
    htmlsrc=urllib.urlopen(myurl).read()
    htmlsrc=htmlsrc.decode('gbk')
    soup = BeautifulSoup(htmlsrc, 'html.parser')
    txtsrc=soup.find_all(id="endText" )
    txtsoup=BeautifulSoup(repr(txtsrc[0]))
    txtstr=txtsoup.get_text()
    txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
    cutstr=cutstring(txtstr)
    tokenstr=nlk.word_tokenize(cutstr)
    for word in tokenstr:
        samplewords.append((category,word))
    print "=",

print ">|"
cfdist=nlk.ConditionalFreqDist(samplewords)
# 知识一词的频率
print cfdist[u'科技'].freq(u'知识')
samplews=[]
# 在教育分类中出现最多的20个词
fd=cfdist[u'教育']
edufd20=sorted(fd.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:20]
for w,c in edufd20:
    print w,">=",c,"||",
    samplews.append(w)
print
# 在科技分类中出现最多的20个词
fd=cfdist[u'科技']
techfd20=sorted(fd.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:20]
for w,c in techfd20:
    print w,">=",c,"||",
    samplews.append(w)
print
samplews=set(samplews)
# 条件频率分布图
cfdist.tabulate(title=u'条件频率分布表',samples=samplews,conditions=[u'科技',u'教育'])
cfdist.plot(title=u'条件频率分布图',samples=samplews,conditions=[u'科技',u'教育'])

```

程序 12-13.py 中有以下几个关键知识点。

### (1) 解析网页

Web 文档通常是 HTML 格式的网页，处理此类文档的关键在于提取标签中的内容，下面的代码片段演示了如何解析网页文本。首先，通过 urllib 模块的 urlopen 打开 Web 链接，并通过 read 函数读取链接指向的 HTML 内容。然后，通过 BeautifulSoup 模块对 HTML 文本



做进一步处理，本例中需要提取新闻的正文，观察新闻的网页文本可发现，其正文通常处于 id 为 endText 的标签内。具体过程为：通过调用 BeautifulSoup 模块的 find\_all 函数找到标签，提取标签中的内容，通过 BeautifulSoup 模块的 get\_text 函数去掉提取内容中残余的 HTML 标记，得到纯净的新闻正文字符串。最后，将新闻正文字符串进行 jieba 分词和 NLTK 分词，以便进行下一步处理。

```
htmlsrc=urllib.urlopen(myurl).read()
htmlsrc=htmlsrc.decode('gbk')
soup = BeautifulSoup(htmlsrc, 'html.parser')
txtsrc=soup.find_all(id="endText" )
txtsoup=BeautifulSoup(repr(txtsrc[0]))
txtstr=txtsoup.get_text()
txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
cutstr=cutstring(txtstr)
tokenstr=nltk.word_tokenize(cutstr)
```

在解析网页时需要特别注意字符编码的问题，中文网页文本的字符编码格式不止 UTF-8 一种，很多中文网页是 GB2312 格式的编码，通过 HTML 的 charset 属性可指定 HTML 文档的字符编码。对于不同字符编码的文本需要调用 decode() 和 encode() 来进行解码和编码。

## (2) 某条件下出现最多的高频词汇

可通过分析某条件下出现频率最多的词汇，近似得到该条件的常用关键词。下面的代码片段对条件频率字典 cfdist 中的教育条件（对于本例而言，此处的条件部分实质上是类别，“某条件下”意味着“某类别下”）进行排序，排序的依据是值（即词汇频数），排序的函数是 Python 的 sorted 函数，返回词频从大到小的词汇列表，列表的前 20 个元素就是出现得最多的高频词汇。

```
# 在教育类 web 文档中出现最多的 20 个词
fd=cfdist[u'教育']
edufd20=sorted(fd.iteritems(),key=lambda x:(x[1],x[0]),reverse=True)[:20]
for w,c in edufd20:
    print w,">=",c,"||",
    samplesw.append(w)
```

程序 12-13.py 的运行结果如下：

```
0.00117233294256
, => 541 || 的 => 356 || 。 => 238 || 在 => 102 || 、=> 86 || 了 => 85 || 是 =>
77 || 学生 => 75 || " => 67 || " => 67 || 学校 => 53 || 毕业生 => 51 || 就业 => 49 ||
% => 48 || 她 => 45 || 也 => 44 || 都 => 43 || 美国 => 40 || 和 => 40 || 有 => 34 ||
, => 126 || 的 => 121 || 。 => 59 || 是 => 38 || 在 => 19 || 了 => 18 || iPhone
=> 17 || 量子 => 15 || 人类 => 15 || " => 14 || " => 14 || 份额 => 13 || : => 12 || 世
界 => 12 || 上 => 12 || 一个 => 12 || Windows => 12 || 机器人 => 11 || 、=> 11 || % =>
11 ||

、 。 的 了 有 上 世界 和 学校 : " " % 份
额 在 是 学生 毕业生 Windows 就业 iPhone , 美国 人类 也 机器人 一
个 她 都 量子
科技 11 59 121 18 5 12 12 9 0 12 14 14 11 13 19
```



```

38      0      0      12      0      17      126      0      15      4      11      12      0      8      15
   教育      86      238      356      85      34      21      1      40      53      33      67      67      48      0      102
77      75      51      0      49      0      541      40      0      44      0      17      45      43      0

```

程序 12-13.py 首先输出“知识”一词在科技分类中的频率（此处的频率不是频数，可理解为“知识”在科技类别的文档中出现的概率）为 0.00117233294256；然后输出在科技、教育类别中出现最多的 20 个高频词。最后，输出高频词的条件频率分布表，并绘制条件频率分布图（如图 12-8 所示），观察图 12-8，较高的曲线为教育，较低的曲线为科技，可见这 20 个高频词属于教育的频数比属于科技的要多。

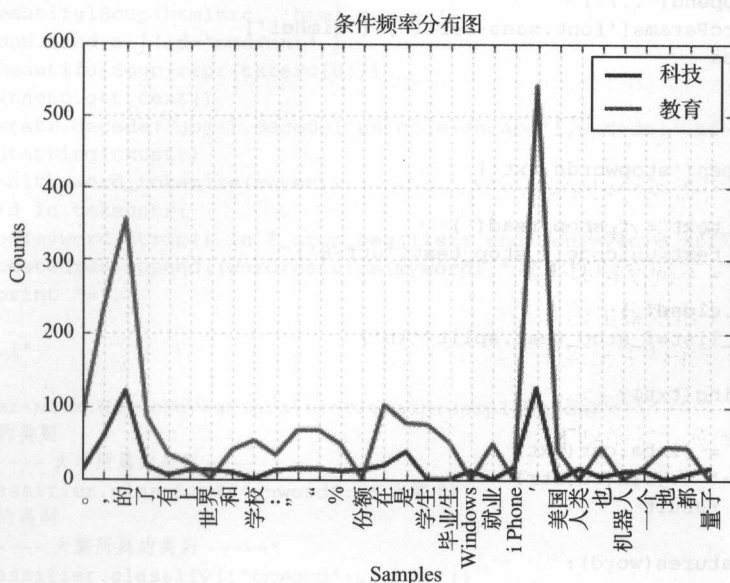


图 12-8 条件频率分布图

### 12.4.6 Web 文档的朴素贝叶斯分类

朴素贝叶斯分类器 (Naive Bayes Classifier, NBC) 发源于古典数学理论，有着坚实的数学基础和稳定的分类效率；同时，NBC 模型所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。

NBC 模型假设属性之间相互独立，通过建立一个属性模型，将相互不独立的属性单独处理。例如对中文文本进行分类的时候，可以建立一个字典来处理一些词组，把一个词组看作一个单独的模式（如果发现特定的问题中存在特殊的模式属性，那么就单独进行处理），这是自然语言与其他分类识别问题的不同点。

#### 1. 词汇特征项分析

假设某样本中的若干文本共分为两类，“中国”这一词汇在两类样本中存在的频率不一样，存在的可能性也不相同。程序 12-14.py 演示了使用 NLTK 分析词条归属类别的可能性：

```

--coding:utf-8--
#code by myhaspl
#12-14.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib
from bs4 import BeautifulSoup

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba

# 停用词字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

def wordfeatures(word):
    return {"cnword":word}

urls=[(u"科技", "http://tech.163.com/16/0203/06/BESLRF50000915BD.html"), (u"科技", "http://tech.163.com/16/0202/01/BEPHEI120009405H.html"), (u"科技", "http://tech.163.com/16/0203/03/BESBB73B000915BD.html"), (u"科技", "http://tech.163.com/16/0203/03/BESAGOPB000915BD.html"), (u"教育", "http://edu.163.com/16/0203/05/BESI2S7500294NE9.html"), (u"教育", "http://kids.163.com/16/0118/06/BDJEMJ3H00294MO6.html"), (u"教育", "http://edu.163.com/16/0128/05/BED4NHBB00294NE9.html"), (u"教育", "http://edu.163.com/16/0202/01/BEPHFQ1800294IIH.html")]
samplewords=[]

print "|=",
for (category,myurl) in urls:
    htmlsrc=urllib.urlopen(myurl).read()
    htmlsrc=htmlsrc.decode('gbk')
    soup = BeautifulSoup(htmlsrc, 'html.parser')
    txtsrc=soup.find_all(id="endText")
    txtsoup=BeautifulSoup(repr(txtsrc[0]))
    txtstr=txtsoup.get_text()
    txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')

```

```

cutstr=cutstring(txtstr)
tokenstr=nlk.word_tokenize(cutstr)
for myword in tokenstr:
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        samplewords.append((wordfeatures(myword),category))
    print "=",

# 测试数据
testurl="http://edu.163.com/16/0204/05/BEV40UHF00294NE9.html"
testwords=[]
htmlsrc=urllib.urlopen(testurl).read()
htmlsrc=htmlsrc.decode('gbk')
soup = BeautifulSoup(htmlsrc, 'html.parser')
txtsrc=soup.find_all(id="endText" )
txtsoup=BeautifulSoup(repr(txtsrc[0]))
txtstr=txtsoup.get_text()
txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
cutstr=cutstring(txtstr)
tokenstr=nlk.word_tokenize(cutstr)
for myword in tokenstr:
    if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        testwords.append((wordfeatures(myword),"教育"))
    print "=",

print "=>|"

classifier=nlk.NaiveBayesClassifier.train(samplewords)
# 大学所属的类别
print u"---- 大学所属的类别 ----"
print classifier.classify({"cnword":u"大学"})
# 大脑所属的类别
print u"---- 大脑所属的类别 ----"
print classifier.classify({"cnword":u"大脑"})
# 测试数据分类的准确率
print nlk.classify.accuracy(classifier,testwords)

# 特征分类最有效的 10 个词
for wf,mostword in classifier.most_informative_features(10):
    print mostword,
print

# 为显示 utf-8, 将 show_most_informative_features 代码进行修改
#classifier.show_most_informative_features(10) 也可直接调用这句, 但是 utf-8 显示有问题
cpdist = classifier._feature_probdist
print('Most Informative Features')

for (fname, fval) in classifier.most_informative_features(10):
    def labelprob(l):
        return cpdist[l, fname].prob(fval)

    labels = sorted([l for l in classifier._labels
                     if fval in cpdist[l, fname].samples()],

```

```

        key=labelprob)
    if len(labels) == 1:
        continue
    l0 = labels[0]
    l1 = labels[-1]
    if cpdist[l0, fname].prob(fval) == 0:
        ratio = 'INF'
    else:
        ratio = '%8.1f' % (cpdist[l1, fname].prob(fval) /
                           cpdist[l0, fname].prob(fval))
    print fname+"="+fval,
    print((' %6s : %6s = %s : 1.0' % (( "%s" % l1)[:6], ("%s" % l0)[:6], ratio)))

```

程序 12-14.py 中有以下几个关键知识点。

1) 去除 Web 文档中的停用词。在处理自然语言数据 (或文本) 之前或之后有时需要自动过滤掉某些字或词, 这些字或词称为 Stop Words (停用词), 停用词主要包括英文字符、数字、数学字符、标点符号及使用频率特高的单汉字等。

下面的代码片段通过 word\_tokenize 完成 NLTK 分词后, 遍历分词结果, 如果含有停用词或是单汉字的, 则作为停用词, 不将其加入最终分词特征列表 (该列表是进一步分析文本特征的依据) 中。

```

tokenstr=nltk.word_tokenize (cutstr)
for myword in tokenstr:
    if not (myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
        samplewords.append ( (wordfeatures (myword),category))
    print "=",

```

2) 构造朴素贝叶斯分类器。通过 nltk.NaiveBayesClassifier 的 train 方法, 将文本特征作为参数, 构造朴素贝叶斯分类器。代码如下所示:

```

classifier=nltk.NaiveBayesClassifier.train(samplewords)

```

3) 某词汇属于某类别的可能性。下面的代码演示了如何计算 “大脑” 一词最可能归属的文本类别:

```

print u"---- 大脑所属的类别 ----"
print classifier.classify(("cnword":u" 大脑 "))

```

4) 对分类最有贡献的词汇。most\_informative\_features 函数将返回对分类最有效的词汇, 参数是词汇数量。代码如下所示:

```

for wf,mostword in classifier.most_informative_features(10):
    print mostword,

```

此外, 还可调用 show\_most\_informative\_features 函数输出对分类有明显作用的词汇具体信息。

5) 测试分类的准确率。

```
# 测试数据分类的准确率
print nltk.classify.accuracy(classifier, testwords)
```

6) 词汇特征分析。NLTK 的朴素贝叶斯分类器对特征项的要求是：每个特征项（在本例中为每个词汇）为一个字典，字典元素的键是特征名称，字典元素的值是特征值，某特征项（在本例中为某词汇）的所有特征组成了该特征项（在本例中为该词汇）的特征字典。代码如下所示：

```
def wordfeatures(word):
    return {"cnword": word}
```

程序 12-14.py 的运行结果如下：

```
|= = = = = ..... = =>|
---- 大学所属的类别 ----
教育
---- 大脑所属的类别 ----
科技
0.977346278317
世界 公司 事先 游戏 之后 领域 采用 学科 里面 技术
Most Informative Features
cnword= 世界      科技 : 教育      =      20.6 : 1.0
cnword= 公司      科技 : 教育      =      12.4 : 1.0
cnword= 事先      科技 : 教育      =       5.8 : 1.0
cnword= 游戏      科技 : 教育      =       5.8 : 1.0
cnword= 之后      科技 : 教育      =       4.5 : 1.0
cnword= 领域      科技 : 教育      =       4.5 : 1.0
cnword= 采用      科技 : 教育      =       4.5 : 1.0
cnword= 学科      科技 : 教育      =       4.1 : 1.0
cnword= 里面      科技 : 教育      =       4.1 : 1.0
cnword= 技术      科技 : 教育      =       4.1 : 1.0
```

观察上述的运行结果，程序 12-14.py 首先分析“大学”最有可能属于教育类别，而“大脑”最有可能属于科技类别；然后输出数据分类的准确率为 0.977346278317，并输出分类最有效的 10 个词汇；最后反映词汇在不同类别中出现的比例，比如，观察输出结果的最后一行，“技术”一词在科技中出现的频率是在教育中出现频率的 4.1 倍。

程序 12-15.py 演示了如何使用 NLTK 对不同词及词长归属类别可能性的分析：

```
#--coding:utf-8--
#code by myhaspl
#12-15.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib
from bs4 import BeautifulSoup
import random
```



```

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba

# 停用词字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

def wordfeatures(word):
    return {"cnword":word,"wcnlen":len(word) }

urls=[(u"科技","http://tech.163.com/16/0203/06/BESLRF50000915BD.html"),(u"科技",
"http://tech.163.com/16/0202/01/BEPHEI120009405H.html"),(u"科技","http://tech.163.
com/16/0203/03/BESBB73B000915BD.html"),(u"科技","http://tech.163.com/16/0203/03/
BESAGOPB000915BD.html"),(u"教育","http://edu.163.com/16/0203/05/BESI2S7500294NE9.
html"),(u"教育","http://kids.163.com/16/0118/06/BDJEMJ3H00294MO6.html"),(u"教育",
"http://edu.163.com/16/0128/05/BED4NHBB00294NE9.html"),(u"教育","http://edu.163.
com/16/0202/01/BEPHFQ1800294IIH.html")]

samplewords=[]

print "|=",
for (category,myurl) in urls:
    htmlsrc=urllib.urlopen(myurl).read()
    htmlsrc=htmlsrc.decode('gbk')
    soup = BeautifulSoup(htmlsrc, 'html.parser')
    txtsrc=soup.find_all(id="endText")
    txtsoup=BeautifulSoup(repr(txtsrc[0]))
    txtstr=txtsoup.get_text()
    txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
    cutstr=cutstring(txtstr)
    tokenstr=nlTK.word_tokenize(cutstr)
    for myword in tokenstr:
        if not(myword.strip() in f_stop_seg_list) and len(myword.strip())>1:
            samplewords.append((myword,category))
    print "=",

# 通过 apply_features 方法返回链表，不在内存中存储所有的特征集，以节约内存
samplelen=len(samplewords)

```

```

trlen=int(samplelen*2/3) # 训练样本的长度
random.shuffle(samplewords)
traintxt=nlTK.classify.apply_features(wordfeatures,samplewords[:trlen])
testtxt=nlTK.classify.apply_features(wordfeatures,samplewords[trlen:])
print ">|"

classifier=nlTK.NaiveBayesClassifier.train(traintxt)
# 大学所属的类别
print u"---- 大学所属的类别 ----"
print classifier.classify({"cnword":u" 大学 ","wcrlen":len(u" 大学 ")})
# 大脑所属的类别
print u"---- 大脑所属的类别 ----"
print classifier.classify({"cnword":u" 大脑 ","wcrlen":len(u" 大脑 ")})
# 测试数据分类的准确率
print nlTK.classify.accuracy(classifier,testtxt)

# 特征分类最有效的10个特征
for wf,mostword in classifier.most_informative_features(10):
    print mostword,
print

# 为显示 utf-8, 将 show_most_informative_features 代码进行修改
#classifier.show_most_informative_features(10) # 也可直接调用这句, 但是 utf-8 显示有问题
cpdist = classifier._feature_probdist
print('Most Informative Features')

for (fname, fval) in classifier.most_informative_features(10):
    def labelprob(l):
        return cpdist[l, fname].prob(fval)

    labels = sorted([l for l in classifier._labels
                     if fval in cpdist[l, fname].samples()],
                    key=labelprob)
    if len(labels) == 1:
        continue
    l0 = labels[0]
    l1 = labels[-1]
    if cpdist[l0, fname].prob(fval) == 0:
        ratio = 'INF'
    else:
        ratio = '%8.1f' % (cpdist[l1, fname].prob(fval)/
                           cpdist[l0, fname].prob(fval))
    print fname,
    print "=",
    print fval,
    print((' %6s : %6s = %s : 1.0' % ((("%s" % l1)[:6], ("%s" % l0)[:6], ratio)))

```

程序 12-15.py 中有以下几个关键知识点。

1) 词汇特征分析。以某词汇的内容和长度作为该词汇的特征项, 代码如下所示:

```

def wordfeatures(word):
    return {"cnword":word,"wcrlen":len(word) }

```

2) 不在内存中存储文本的所有特征项集, 以节约内存。通过 `apply_features` 方法, 创建文本特征项集的列表, 可不在内存中存储所有特征项集, 当特征项数量特别大时, 这种方法能节约内存。代码如下所示:

```
traintxt=nltk.classify.apply_features(wordfeatures,samplewords[:trlen])
testtxt=nltk.classify.apply_features(wordfeatures,samplewords[trlen:])
```

程序 12-15.py 的运行结果如下:

```
|===== =>|
---- 大学所属的类别 ----
教育
---- 大脑所属的类别 ----
科技
0.863354037267
8 世界 公司 7 信息 领域 6 学科 里面 事先
Most Informative Features
wcrlen = 8          科技 : 教育      = 15.1 : 1.0
cnword = 世界       科技 : 教育      = 11.8 : 1.0
cnword = 公司       科技 : 教育      = 10.2 : 1.0
wcrlen = 7          科技 : 教育      = 5.6 : 1.0
cnword = 信息       科技 : 教育      = 5.5 : 1.0
cnword = 领域       科技 : 教育      = 5.5 : 1.0
wcrlen = 6          科技 : 教育      = 4.8 : 1.0
cnword = 学科       科技 : 教育      = 3.9 : 1.0
cnword = 里面       科技 : 教育      = 3.9 : 1.0
cnword = 事先       科技 : 教育      = 3.9 : 1.0
```

观察上述运行结果, 程序 12-15.py 首先分析“大学”最有可能属于教育类别, 而“大脑”最有可能属于科技类别; 然后输出数据分类准确率为 0.863354037267, 并输出分类最有效的 10 个特征, 用前 3 个特征 8、“世界”、“公司”为例进行剖析可以看出, 长度为 8、“世界”、“公司”是对分类最有效的前 3 个特征; 最后反映特征在不同类别中出现的比例, 比如, 观察输出结果的最后一行, “事先”在科技类别中出现的频率是教育类中出现频率的 3.9 倍。

程序 12-16.py 演示了如何使用 NLTK 的不同词归属类别可能性的分析, 与 12-14.py 的不同之处在于: 特征字典的值为该词汇是否出现。

```
#!/usr/bin/env python
#coding:utf-8--
#code by myhaspl
#12-16.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib
from bs4 import BeautifulSoup
import random

import sys
```

```
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba
```

```
def cutstring(txt):
```

```
    # 分词
```

```
    cutstr = jieba.cut(txt)
```

```
    result=" ".join(cutstr)
```

```
    return result
```

```
def txtfeatures(allwords, tokenw, stopseg):
```

```
    features={}
```

```
    for myword in set(allwords):
```

```
        if not(myword.strip() in stopseg) and len(myword.strip())>1:
```

```
            features["cnword-"+myword]=(myword in set(tokenw))
```

```
            print "=",
```

```
    return features
```

```
# 停用词字典
```

```
f_stop = open('stopwords.txt')
```

```
try:
```

```
    f_stop_text = f_stop.read()
```

```
    f_stop_text=unicode(f_stop_text, 'utf-8')
```

```
finally:
```

```
    f_stop.close()
```

```
f_stop_seg_list=f_stop_text.split('\n')
```

```
urls=[(u"科技", "http://tech.163.com/16/0203/06/BESLRF50000915BD.html"), (u"科技", "http://tech.163.com/16/0202/01/BEPHEI120009405H.html"), (u"科技", "http://tech.163.com/16/0203/03/BESBB73B000915BD.html"), (u"科技", "http://tech.163.com/16/0203/03/BESAGOPB000915BD.html"), (u"教育", "http://edu.163.com/16/0203/05/BESI2S7500294NE9.html"), (u"教育", "http://kids.163.com/16/0118/06/BDJEMJ3H00294MO6.html"), (u"教育", "http://edu.163.com/16/0128/05/BED4NHBB00294NE9.html"), (u"教育", "http://edu.163.com/16/0202/01/BEPHFQ1800294IIH.html")]
```

```
samplewords=[]
```

```
allw=[]
```

```
tokendocstr=[]
```

```
print "|=",
```

```
for (category, myurl) in urls:
```

```
    htmlsrc=urllib.urlopen(myurl).read()
```

```
    htmlsrc=htmlsrc.decode('gbk')
```

```
    soup = BeautifulSoup(htmlsrc, 'html.parser')
```

```
    txtsrc=soup.find_all(id="endText")
```

```
    txtsoup=BeautifulSoup(repr(txtsrc[0]))
```

```
    txtstr=txtsoup.get_text()
```

```
    txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
```

```
    cutstr=cutstring(txtstr)
```

```
    tokendocstr.append(nltk.word_tokenize(cutstr))
```

```

for docstr in tokendocstr:
    for w in docstr:
        allw.append(w)
allw=set(allw)

samplefeatures=[]
i=0
for docstr in tokendocstr:
    docfeature=txtfeatures(allw,docstr,f_stop_seg_list)
    samplefeatures.append((docfeature,urls[i][0]))
    i=i+1

samplelen=len(samplefeatures)
trlen=int(samplelen*2/3)# 训练样本长度
random.shuffle(samplefeatures)

traintxt=samplefeatures[:trlen]
testtxt=samplefeatures[trlen:]
print "=>|"

classifier=nltk.NaiveBayesClassifier.train(traintxt)
# 测试数据分类准确率
print nltk.classify.accuracy(classifier,testtxt)
classifier.show_most_informative_features(15)

```

观察程序 12-16.py, 对文本进行特征分析时, 其文本特征项由非停用词特征组成, 每个非停用词的特征计算方式为: 特征名称为非停用词内容, 特征值为 True (该非停用词在该文本中出现) 或 False (该非停用词未在该文本中出现)。代码如下所示:

```

def txtfeatures(allwords,tokenw,stopseg):
    features={}
    for myword in set(allwords):
        if not(myword.strip() in stopseg) and len(myword.strip())>1:
            features["cnword-"+myword]=(myword in set(tokenw))
            print "=",
    return features

```

程序 12-16.py 首先输出分类准确率为 0.666666666667; 然后输出对分类最有效的前 15 个特征。程序 12-16.py 的运行结果如下:

```

.....= =>|
0.666666666667
Most Informative Features
cnword- 国外 = False      科技 : 教育 = 2.2 : 1.0
cnword- 一年 = False     科技 : 教育 = 2.2 : 1.0
.....
cnword- 厕所 = False     科技 : 教育 = 2.2 : 1.0
cnword- 确实 = False     科技 : 教育 = 2.2 : 1.0

```

## 2. 网页分类

程序 12-17.py 演示了如何使用 NLTK 对未知网页进行朴素贝叶斯分类:



```

--coding:utf-8--
#code by myhaspl
#12-17.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib
from bs4 import BeautifulSoup

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']
import jieba

def cutstring(txt):
    # 分词
    cutstr = jieba.cut(txt)
    result=" ".join(cutstr)
    return result

def txtfeatures(allwords,tokenw,stopseg):
    features={}
    for myword in set(allwords):
        if not(myword.strip() in stopseg) and len(myword.strip())>1:
            features["cnword-"+myword]=(myword in set(tokenw))
    return features

# 停用词字典
f_stop = open('stopwords.txt')
try:
    f_stop_text = f_stop.read()
    f_stop_text=unicode(f_stop_text,'utf-8')
finally:
    f_stop.close()
f_stop_seg_list=f_stop_text.split('\n')

urls=[(u"科技", "http://tech.163.com/16/0203/06/BESLRF50000915BD.html"),(u"科技",
"http://tech.163.com/16/0202/01/BEPHEI120009405H.html"),(u"科技", "http://tech.163.
com/16/0203/03/BESBB73B000915BD.html"),(u"科技", "http://tech.163.com/16/0203/03/
BESAGOPB000915BD.html"),(u"教育", "http://edu.163.com/16/0203/05/BESI2S7500294NE9.
html"),(u"教育", "http://kids.163.com/16/0118/06/BDJEMJ3H00294MO6.html"),(u"教育",
"http://edu.163.com/16/0128/05/BED4NHBB00294NE9.html"),(u"教育", "http://edu.163.
com/16/0202/01/BEPHFQ1800294IIH.html")]

samplewords=[]
allw=[]

```

```

tokendocstr=[]
print "|=",
for (category,myurl) in urls:
    htmlsrc=urllib.urlopen(myurl).read()
    htmlsrc=htmlsrc.decode('gbk')
    soup = BeautifulSoup(htmlsrc, 'html.parser')
    txtsrc=soup.find_all(id="endText" )
    txtsoup=BeautifulSoup(repr(txtsrc[0]))
    txtstr=txtsoup.get_text()
    txtstr=txtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
    cutstr=cutstring(txtstr)
    tokendocstr.append(nltk.word_tokenize(cutstr))
    print "=",

for docstr in tokendocstr:
    for w in docstr:
        allw.append(w)
allw=set(allw)

samplefeatures=[]
i=0
for docstr in tokendocstr:
    docfeature=txtfeatures(allw,docstr,f_stop_seg_list)
    samplefeatures.append((docfeature,urls[i][0]))
    i=i+1

traintxt=samplefeatures

print "=>|"

classifier=nltk.NaiveBayesClassifier.train(traintxt)

# 新的网页
testmyurl="http://tech.163.com/16/0207/09/BF7AR5D4000915BD.html"
testhtmlsrc=urllib.urlopen(testmyurl).read()
testhtmlsrc=testhtmlsrc.decode('gbk')
soup = BeautifulSoup(testhtmlsrc, 'html.parser')
testtxtsrc=soup.find_all(id="endText" )
testtxtsoup=BeautifulSoup(repr(testtxtsrc[0]))
testtxtstr=testtxtsoup.get_text()
testtxtstr=testtxtstr.decode('gbk').decode("unicode-escape").encode('utf-8')
testcutstr=cutstring(testtxtstr)
testtokendocstr=nltk.word_tokenize(testcutstr)
testdocfeature=txtfeatures(allw,testtokendocstr,f_stop_seg_list)
# 测试数据分类
print classifier.classify(testdocfeature)

```

程序 12-17.py 中有以下关键知识点。

1) 文本特征分析。对文本特征的分析首先从词汇特征入手，在去除非停用词后，余下

的词汇对文本特征的提取均有意义，每个非停用词的特征名称均为非停用词的内容，特征值为 True（该非停用词在该文本中出现）或 False（该非停用词未在该文本中出现）。

```
def txtfeatures(allwords, tokenw, stopseg):
    features={}
    for myword in set(allwords):
        if not(myword.strip() in stopseg) and len(myword.strip())>1:
            features["cnword-"+myword]=(myword in set(tokenw))
    return features
```

2) 分类结果。调用分类器对象 classifier 的 classify 方法对未知样本数据进行分类：

```
print classifier.classify(testdocfeature)
```

程序 12-17.py 的运行结果如下：

```
..... == =>|
科技
```

观察 12-17.py 的运行结果可以得知，链接 <http://tech.163.com/16/0207/09/BF7AR5D4000915BD.html> 指向的 Web 文档为科技类别，在浏览器中打开该页面，标题为“伦敦希望谷歌能将无人驾驶汽车测试带进英国”，该文档确实属于科技类别，分类准确无误。

## 12.4.7 语法结构分析

语法是语言学的一个分支，研究按确定用法来运用的词类、词的曲折变化或表示相互关系的其他手段及词在句子中的功能和关系。包含词的构词、构形的规则和组词成句的规则。

程序 12-18.py 演示了如何使用 NLTK 识别名词短语及语法树分析：

```
--coding:utf-8--
#code by myhaspl
#12-18.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk

import sys
sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']

from jieba import posseg

def cutstrpos(txt):
    # 分词 + 词性
    cutstr = posseg.cut(txt)
    result=""
    for word, flag in cutstr:
```

```

        result+=word+"/"+flag+' '
    return result

```

```

sentencestr=[]

```

```

txtstr=""

```

据美国华盛顿自由灯塔网站 2 月 17 日报道, 政府问责局 16 日发布的这份报告称, 美国导弹防御局继续将大笔资金用于未被证明能够应对来自伊朗或朝鲜的弹道导弹袭击的技术。

```

.....
"""

```

```

# 生成 nltk 格式的词性单词集

```

```

posstr=cutstrpos(txtstr)

```

```

strtag=[nltk.tag.str2tuple(word) for word in posstr.split()]

```

```

# 句子分割

```

```

sentstr=[]

```

```

for wtag in strtag:

```

```

    if wtag[0].strip() in [",", ".", "!", "?", ":", ";"]:

```

```

        sentencestr.append(sentstr)

```

```

        sentstr=[]

```

```

    else:

```

```

        if wtag[1]!="X":

```

```

            sentstr.append((wtag[0].strip(),wtag[1]))

```

```

# 输出分割后的句子

```

```

for wordstr in sentencestr:

```

```

    print

```

```

    for word in wordstr:

```

```

        print word[0],"/",word[1],

```

```

# 名词短语识别

```

```

grammar=r"""

```

```

NP: { (<A>|<R>|<MQ>)*<N>+<K>*}

```

```

NP: { (<A>|<R>|<MQ>)*<NS>+<K>*}

```

```

NP: {<M>+}

```

```

"""

```

```

cp=nltk.RegexpParser(grammar)

```

```

result1=cp.parse(sentencestr[0])

```

```

result1.draw()

```

```

result2=cp.parse(sentencestr[1])

```

```

result2.draw()

```

程序 12-18.py 中有以下关键知识点。

### (1) 生成 NLTK 格式的词性单词集

首先, 通过 jieba 中文分词组件分词并标注词性, 分词结果为形如“词汇 1/ 词性 1 词汇 2/ 词性 2.../... 词汇 n/ 词性 n”的字符串, 每个词汇之间用空格进行分隔; 然后, 调用 nltk.tag 的 str2tuple 函数生成 NLTK 格式的词典单词集。

```

posstr=cutstrpos(txtstr)

```

```


strtag=[nltk.tag.str2tuple(word) for word in posstr.split()]

```

## (2) 句子分割并去除标点

在文本字符串中查找“,”、“。”等常用的中文标点符号,将文本分割成句子;此外,将词性是“X”类别(这个词性类别通常为标点)的词删除。


```
for wtag in strtag:
    if wtag[0].strip() in [",", ".", "!", "?", ":", ";"]:
        sentencestr.append(sentstr)
        sentstr=[]
    else:
        if wtag[1]!="X":
            sentstr.append((wtag[0].strip(),wtag[1]))
```

 **提示** jieba 中文分词采用的是和 ictclas 兼容的标记法。某词的词性“X”表示它是非语素字,非语素字只是一个符号和未知数。

## (3) 名词短语识别

使用 NLTK 识别语法结构,最好的方式是指定某语法结构的正则表达式。为简单起见,12-18.py 根据本例的实际情况,将常用的名词短语格式定义为“形容词/代词/数量词+名词(包括地名)若干+后缀”或“日期”的格式。

```
grammar=r"""
NP: { (<A>|<R>|<MQ>)*<N>+<K>*}
NP: { (<A>|<R>|<MQ>)*<NS>+<K>*}
NP: {<M>+}
"""
```

 **提示** A 表示形容词, R 表示代词, MQ 表示数量词, M 表示日期词, K 表示后缀。具体请参见 ICTCLAS 汉语词性标注集。

## (4) 解析语法结构,并绘制语法结构图

首先通过 RegexpParser 来构造语法结构解析器,然后,通过解析器对象的 parse 方法解析语法结构,并通过 draw 方法绘图。

```
cp=nltk.RegexpParser(grammar)
result1=cp.parse(sentencestr[0])
result1.draw()"""
```

程序 12-18.py 运行后,首先输出句子分割结果,如下所示。然后输出语法结构图,如图 12-9 所示。

```
据 / P 美国 / NS 华盛顿 / NS 自由 / A 灯塔 / N 网站 / N 2 / M 月 / M 17 / M 日 / M
报道 / V
政府 / N 问责局 / N 16 / M 日 / M 发布 / V 的 / UJ 这份 / MQ 报告 / N 称 / V
.....
因为 / C 还 / D 需要 / V 进行 / V 大规模 / B 的 / UJ 更新 / D
```



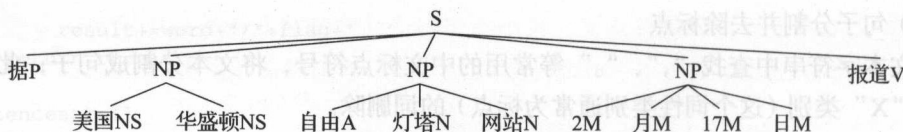


图 12-9 语法结构图

观察图 12-9 所示的语法结构图，可看出程序 12-18.py 识别出名词短语 NP（比如：“美国华盛顿”），并将它们单独绘制在树的第三层。此外，树的第一层是 S，表示句子，第二层是该句的语法结构：P 表示介词，NP 表示名词短语，V 表示动词。

### 12.4.8 Web 文档聚类

文档聚类（Text Clustering）主要是依据著名的聚类假设：同类文档的相似度较大，而不同类文档的相似度较小。作为一种无监督的机器学习方法，聚类由于不需要训练过程，以及不需要预先对文档进行手工标注类别，因此具有一定的灵活性和较高的自动化处理能力，已经成为对文本信息进行有效地组织、摘要和导航的一种重要手段。

#### 1. 重点词聚类

程序 12-19.py 演示了如何使用 NLTK 通过电影评论对电影进行聚类。对于评论而言，重点词一般是情感类及相关词汇，本例将重点词汇定义为名词、形容词、语气词及叹词。

```

#--coding:utf-8--
#code by myhaspl
#12-19.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib

import numpy
import re
import sys

sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']

from jieba import posseg

def cutstrpos(txt):
    # 分词 + 词性
    cutstr = posseg.cut(txt)
    result=""
    for word, flag in cutstr:
        result+=word+"/"+flag+' '
    return result
  
```

```

def inittxtw(allwords):
    txtwords={}
    for w in allwords:
        txtwords[w]=0
    return txtwords

def txtfeatures(allwdt,tokenw):
    # 词频特征提取
    txtfd=nlTK.FreqDist(tokenw)
    for key,val in sorted(txtfd.iteritems()):
        allwdt[key]=val
    return allwdt.values()

# 根据电影评论将电影聚类
urls=[(u"功夫熊猫3","http://ent.163.com/16/0129/00/BEF5L97P00034R77.html"),(u"唐人街探案","http://ent.163.com/15/1230/15/BC3GSVQ700034R77.html"),(u"恶棍天使","http://ent.163.com/15/1224/07/BBJ60V1H00034R77.html"),(u"老炮儿","http://ent.163.com/15/1224/00/BBIED1QL00034R77.html"),(u"寻龙诀","http://ent.163.com/15/1218/03/BB3A3I2B00034R77.html"),(u"万万没想到","http://ent.163.com/15/1218/03/BB3ACSP700034R77.html"),(u"星球大战7","http://ent.163.com/15/1217/15/BB21INTG00034R77.html"),(u"饥饿游戏3(下)","http://ent.163.com/15/1120/07/B8RKKR9C00034R77.html"),(u"九层妖塔","http://ent.163.com/15/0930/01/B4NN919N00034R77.html"),(u"超能查派","http://ent.163.com/15/0508/06/AP2T6CP400034R77.html"),(u"我是路人甲","http://ent.163.com/15/0702/17/ATHL56DT00034R77.html"),(u"港囧","http://ent.163.com/15/0925/05/B4BA6IJ600034R77.html"),(u"007: 幽灵党","http://ent.163.com/15/1113/08/B89PJSGE00034R77.html"),(u"小黄人大眼萌","http://ent.163.com/15/0913/07/B3CI689C00034R77.html")]

allw=[]
tokendocstr=[]
i=0
errorfilm=[]
for (film,myurl) in urls:
    print "\n***** 《"+film+"》 核心词 *****"
    htmlsrc=urllib.urlopen(myurl).read()
    htmlsrc=htmlsrc.decode('gbk')
    pattern=re.compile(r'((class="end-text">)|(<br /></p><p>))(.*<style>.*--></p>)*(.*?)(</p></div>|<div id=)')
    match = pattern.findall(repr(htmlsrc))
    matchstr=""
    if match:
        mstr=match[0][4].encode('utf-8').decode("unicode-escape")
        matchstr+=mstr
    else:
        print film,"无法取得评论!"
        errorfilm.append(i)
    i+=1
    txtstr=matchstr.replace("</p>","").replace("<p>","").replace("<span>","").replace("</

```

```
# 生成 nltk 格式的词性单词集
posstr=cutstrpos(txtstr)
strtag=[nltk.tag.str2tuple(word) for word in posstr.split()]
cutstr=""
for word,tag in strtag:
    # 只处理名词、形容词、语气词、叹词
    if tag[0]=="N" or tag[0]=="A" or tag[0]=="Y" or tag[0]=="E":
        cutstr+=word+" "
print cutstr
tokenocstr.append(nltk.word.tokenize(cutstr))
```

```
docstr in tokendocstr:
    for w in docstr:
        allw.append(w)
w=set(allw)

plefeatures=[]

docstr in tokendocstr:
    allwdt=inittxtw(allw)
    docfeature=numpy.array([txtfeatures(allwdt,docstr)])
    samplefeatures.append(docfeature)
```

```

AC 聚类
nt u"-----GAAC 聚类-----"
classifier=nltk.cluster.gaac.GAACClusterer(num_clusters=5,normalise=True)
classifier.cluster(vectors=samplefeatures)

data in samplefeatures:
print urls[i][0]," 聚类编号:"
print txtclassifier.classify(data)
i+=1

eans 聚类
nt u"-----kmeans 聚类-----"
classifier=nltk.cluster.kmeans.KMeansClusterer(num_means=5,distance=nltk.
util.euclidean_distance)
classifier.cluster(samplefeatures)

data in samplefeatures:
print urls[i][0]," 聚类编号:"
print txtclassifier.classify(data)
i+=1

```

程序 12-19.py 中有以下关键知识点。

### (1) 文本特征提取

通过提取重点词在文本中出现的数量作为该文本的特征，代码如下所示：

```
def txtfeatures(allwdt,tokenw):
    # 词频特征提取
    txtfd=nlTK.FreqDist(tokenw)
    for key,val in sorted(txtfd.iteritems()):
        allwdt[key]=val
    return allwdt.values()
```

### (2) 重点词筛选

N 表示名词，A 表示形容词，Y 表示语气词，E 表示叹词，将这 4 类词筛选出来作为评论的重点词，参与文本特征提取计算。代码如下所示：

```
for word,tag in strtag:
    # 只处理名词、形容词、语气词、叹词
    if tag[0]=="N" or tag[0]=="A" or tag[0]=="Y" or tag[0]=="E":
        cutstr+=word+" "
```

### (3) GAA 算法

群平均聚类，简称 GAA (The Group Average Agglomerative)，核心思想是：开始以每一个向量作为单个群，然后迭代，将有最近的质心的集全部成群，这一过程持续到仅有一个群，这种合并产生了树状图。纵观这一过程可以看出，GAA 属于层次式聚类方法，它的基本步骤如下：

1) 将每个对象归为一类，共得到  $N$  类，每类仅包含一个对象。类与类之间的距离就是它们所包含的对象之间的距离。

2) 找到最接近的两个类并合并成一个类，于是类的总数少了一个。

3) 重新计算新的类与所有旧类之间的距离。

4) 重复第 2 步和第 3 步，直到最后合并成一个类为止（此类包含了  $N$  个对象）。

整个聚类过程其实是建立了 GAA 树（如图 12-10 所示）。关键是如何判断两个类之间的相似度，对于文本而言，一般使用余弦距离来判断相似性。

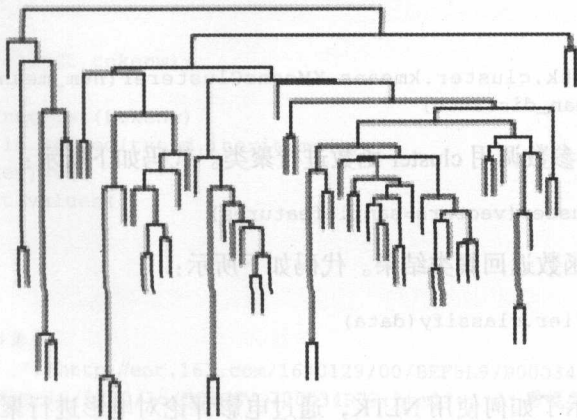


图 12-10 GAA 树

使用 NLTK 进行 GAA 的过程一般如下:

1) 通过 `nltk.cluster.gaac` 的 `GAAClusterer` 函数构造 GAA 对象, 其中 `normalise` 参数为 `True` 时表示使用余弦相似度。代码如下所示:

```
txtclassifier=nltk.cluster.gaac.GAAClusterer(num_clusters=5,normalise=True)
```

2) 以聚类样本为参数调用 `cluster` 函数进行聚类。代码如下所示:

```
txtclassifier.cluster(vectors=samplefeatures)
```

3) 通过 `classify` 函数返回聚类结果。代码如下所示:

```
print txtclassifier.classify(data)
```

(4) K 均值算法

K 均值算法以欧式距离作为相似度来测度, 它是求对应某一初始聚类中心向量  $V$  的最优分类, 使得评价指标  $J$  最小。算法采用误差平方和准则函数作为聚类准则函数, 即认为两个对象的距离越近, 其相似度就越大, 因此算法把产生紧凑且独立的簇作为最终目标。

该算法在每次迭代中对数据集中剩余的每个对象, 根据其与其他各个簇中心的距离将每个对象重新赋给最近的簇。当考察完所有的数据对象后, 一次迭代运算完成, 新的聚类中心被计算出来。如果在一次迭代前后, 评价指标  $J$  的值没有发生变化, 就说明算法已经收敛。

算法的具体过程如下:

1) 从  $N$  个文档随机选取  $K$  个文档作为质心。

2) 对剩余的每个文档测量其到每个质心的距离, 并把它归到最近的质心的类。

3) 重新计算已经得到的各个类的质心。

4) 迭代第 2 步和第 3 步直至新的质心与原质心相等或小于指定阈值, 算法结束。

使用 NLTK 进行 K 均值算法的过程一般如下:

1) 通过 `nltk.cluster.kmeans` 的 `KMeansClusterer` 函数构造 K 均值聚类器对象。代码如下所示:

```
txtclassifier=nltk.cluster.kmeans.KMeansClusterer(num_means=5,distance=nltk.cluster.util.euclidean_distance)
```

2) 以聚类样本为参数调用 `cluster` 函数进行聚类。代码如下所示:

```
txtclassifier.cluster(vectors=samplefeatures)
```

3) 通过 `classify` 函数返回聚类结果。代码如下所示:

```
print txtclassifier.classify(data)
```

## 2. 关键词聚类

程序 12-20.py 演示了如何使用 NLTK, 通过电影评论对电影进行聚类。与 12-19.py 不同



的是,本例聚类的依据是关键词,计算关键词的算法为TF/IDF权重算法,取权重最大的前30个词作为关键词。代码如下所示:

```
#--coding:utf-8--
#code by myhaspl
#12-20.py
from __future__ import unicode_literals
from __future__ import division
import pylab
import nltk
import urllib

import numpy
import re
import sys

import jieba.analyse

sys.path.append("../")
pylab.mpl.rcParams['font.sans-serif']=['SimHei']

from jieba import posseg

def cutstrpos(txt):
    # 分词 + 词性
    cutstr = posseg.cut(txt)
    result=""
    for word, flag in cutstr:
        result+=word+"/"+flag+' '
    return result

def inittxtw(allwords):
    txtwords={}
    for w in allwords:
        txtwords[w]=0
    return txtwords

def txtfeatures(allwdt,tokenw):
    # 统计词频数
    txtfd=nltk.FreqDist(tokenw)
    for key,val in sorted(txtfd.iteritems()):
        allwdt[key]=val
    return allwdt.values()
```

程序 12-20.py 首先输出各部电影评论的关键词,然后输出电影的聚类结果。聚类结果如下:

```
# 根据电影评论将电影聚类
urls=[(u"功夫熊猫3","http://ent.163.com/16/0129/00/BEF5L97P00034R77.html"),(u"唐人街探案","http://ent.163.com/15/1230/15/BC3GSVQ700034R77.html"),(u"恶棍天使","http://ent.163.
```

```
com/15/1224/07/BBJ60V1H00034R77.html"),(u"老炮儿","http://ent.163.com/15/1224/00/
BBIED1QL00034R77.html"),(u"寻龙诀","http://ent.163.com/15/1218/03/BB3A3I2B00034R77.
html"),(u"万万没想到","http://ent.163.com/15/1218/03/BB3ACSP700034R77.html"),(u"星球
大战7","http://ent.163.com/15/1217/15/BB21INTG00034R77.html"),(u"饥饿游戏3(下)
","http://ent.163.com/15/1120/07/B8RKKR9C00034R77.html"),(u"九层妖塔","http://ent.163.
com/15/0930/01/B4NN919N00034R77.html"),(u"超能查派","http://ent.163.com/15/0508/06/
AP2T6CP400034R77.html"),(u"我是路人甲","http://ent.163.com/15/0702/17/ATHL56DT00034R77.
html"),(u"港囧","http://ent.163.com/15/0925/05/B4BA6IJ600034R77.html"),(u"007: 幽灵党
","http://ent.163.com/15/1113/08/B89PJSGE00034R77.html"),(u"小黄人大眼萌","http://ent.163.
com/15/0913/07/B3CI689C00034R77.html")]
```

```
allw=[]
tokendocstr=[]
i=0
errorfilm=[]
for (film,myurl) in urls:
    print "\n*****《"+film+"》关键词*****"
    htmlsrc=urllib.urlopen(myurl).read()
    htmlsrc=htmlsrc.decode('gbk')
    pattern=re.compile(r'((class="end-text">)|(<br /></p><p>))(.*)<style>.*--></
p>)*(.(?<div id="')
    match = pattern.findall(repr(htmlsrc))
    matchstr=""
    if match:
        mstr=match[0][4].encode('utf-8').decode("unicode-escape")
        matchstr+=mstr
    else:
        print film,"无法取得评论!"
        errorfilm.append(i)
    i+=1

    txtstr=matchstr.replace("</p>","").replace("<p>","").replace("<span>","").replace("</
span>","").replace(r'<b style="mso-bidi-font-weight: normal">','').replace(r'<b>','')

    # 提取前 30 位 TF/IDF 权重最大的关键词
    topK = 30
    tags = jieba.analyse.extract_tags(txtstr, topK=topK)
    cutstr=" ".join(tags)
    print cutstr
    tokendocstr.append(nltk.word_tokenize(cutstr))

for docstr in tokendocstr:
    for w in docstr:
        allw.append(w)
allw=set(allw)

samplefeatures=[]
```

```

for docstr in tokendocstr:
    allwdt=inittxtw(allw)
    docfeature=numpy.array(txtfeatures(allwdt,docstr))
    samplefeatures.append(docfeature)

#GAAC 聚类
print u"-----GAAC 聚类 -----"
txtclassifier=nltk.cluster.gaac.GAACClusterer(num_clusters=5,normalise=True)

txtclassifier.cluster(vectors=samplefeatures)
i=0
for data in samplefeatures:
    print urls[i][0]," 聚类编号 : "
    print txtclassifier.classify(data)
    i+=1

#kmeans 聚类
print u"-----kmeans 聚类 -----"
txtclassifier=nltk.cluster.kmeans.KMeansClusterer(num_means=5,distance=nltk.
cluster.util.euclidean_distance)
txtclassifier.cluster(samplefeatures)
i=0
for data in samplefeatures:
    print urls[i][0]," 聚类编号 : "
    print txtclassifier.classify(data)
    i+=1

```

程序 12-20.py 的核心在于提取关键词，本例基于 TF/IDF 权重，即：取 TF/IDF 权重最高的前 30 个词作为关键词。

TF-IDF 是一种统计方法，用以评估某一字词对于一个文件集或一个语料库中的某一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF 指词频 (Term Frequency)，IDF 指逆向文件频率 (Inverse Document Frequency)。TF 的意义在于可表示词条  $t$  在文档  $d$  中出现的频率，IDF 的意义在于如果包含词条  $t$  的文档越少，也就是  $n$  越小，IDF 越大，则说明词条  $t$  具有很好的类别区分能力。

下面的代码片断通过调用 jieba 分词组件的 analyse.extract\_tags 方法来分析 TF/IDF 权重：

```

# 提取前 30 位 TF/IDF 权重最大的关键词
topK = 30
tags = jieba.analyse.extract_tags(txtstr, topK=topK)
cutstr=" ".join(tags)-
print cutstr
tokendocstr.append(nltk.word_tokenize(cutstr))

```

程序 12-20.py 首先输出各部电影评论的关键词，然后输出电影的聚类结果，聚类编号相同者为观众心目中的同一类型电影。运行结果如下：

\*\*\*\*\* 《功夫熊猫 3》关键词 \*\*\*\*\*

熊猫 功夫 阿宝 天煞 梦工厂 影片 反派 动画电影 修炼 师父 电影 搞笑 动画 角色  
 残豹 前作 浣熊 无厘头 一集 顿悟 气功 一部 没有 包子 剧情 显得 武功 孔雀 觉醒 以及  
 .....

-----GAAC 聚类-----

功夫熊猫 3 聚类编号:

1

唐人街探案 聚类编号:

2

.....

-----kmeans 聚类-----

功夫熊猫 3 聚类编号:

4

唐人街探案 聚类编号:

1

.....

## 12.5 小结

现在已经是数字时代了,可获得的需要处理的文本数量越来越多,而文本数据与其他数据不同,它具有非结构化的特点,这对文本数据的分类与自然语言的处理提出了更高的要求。

本章首先介绍了基于余弦相似度、朴素贝叶斯算法的文本分类及 Web 文档分类技术,并详细讲解了文本预处理技术(中文分词、停用词处理等)、文本特征提取技术等内容。然后阐述了使用 NLTK 工具对中文进行自然语言处理的技术,包括分词与词性标注、词汇特征指标、Web 文档分析、Web 文档分类及聚类、文本语法结构分析等内容。

## 思考题

(1) 基于余弦相似度算法对新闻网页进行分类,观察其效果。

(2) 基于 SVM 算法对新闻网页进行分类,观察其效果。



提示 文本样本的特征组可以定义为其包含的词条在某类别中出现的先验概率。

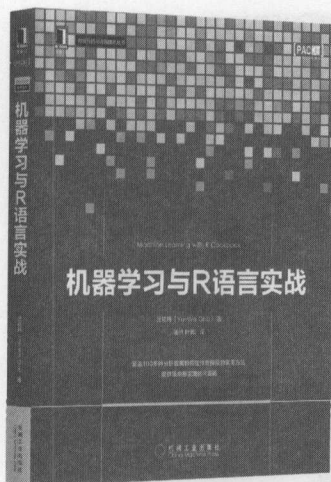
(3) 基于加权朴素贝叶斯算法对新闻网页进行分类,观察其效果。



提示 可以以新闻的标题为关键句,将标题中包含的词赋予较大的权重,将权重直接乘以先验概率作为标题词条的先验概率。

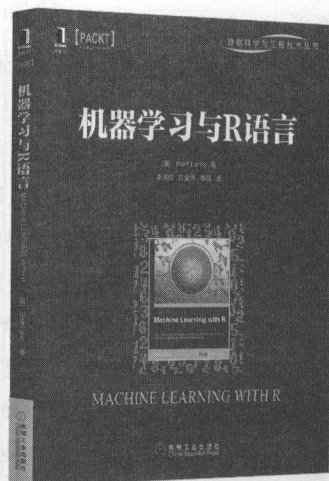


## 推荐阅读



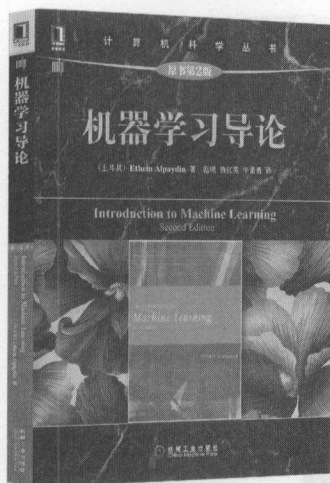
### 机器学习与R语言实战

作者：丘祐玮 (Yu-Wei Chiu) 译者：潘怡 等  
ISBN: 978-7-111-53595-9 定价：69.00元



### 机器学习与R语言

作者：Brett Lantz 译者：李洪成 等  
ISBN: 978-7-111-49157-6 定价：69.00元



### 机器学习导论 (原书第3版)

作者：埃塞姆·阿培丁 译者：范明  
ISBN: 978-7-111-52194-5 定价：79.00元

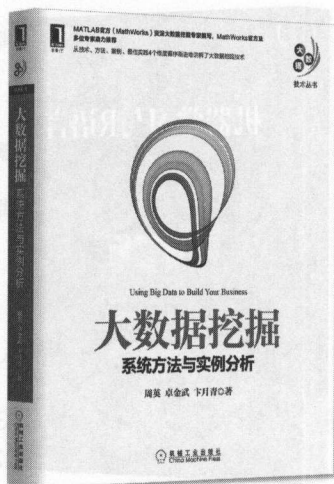


### 机器学习：实用案例解析

作者：Drew Conway 等 译者：陈开江 等  
ISBN: 978-7-111-41731-6 定价：69.00元



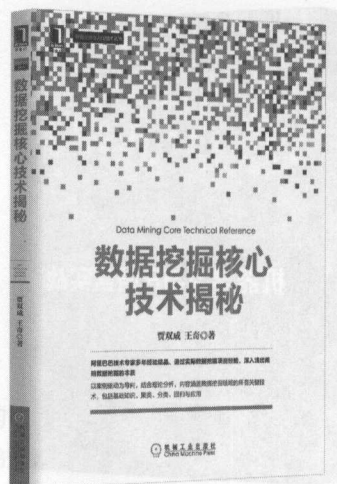
## 推荐阅读



### 大数据挖掘：系统方法与实例分析

作者：周英 卓金武 卞月青

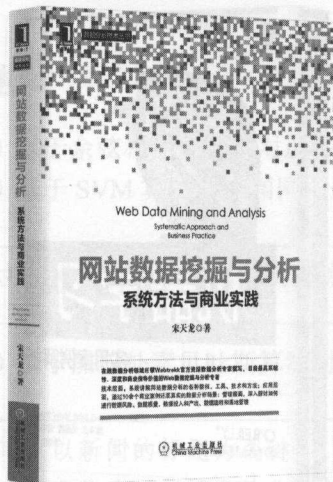
ISBN: 978-7-111-53267-5 定价：79.00元



### 数据挖掘核心技术揭秘

作者：贾双成 王奇

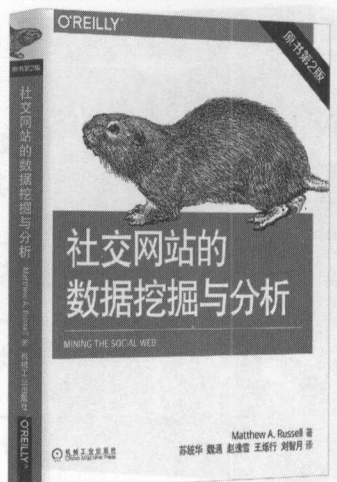
ISBN: 978-7-111-51924-9 定价：59.00元



### 网站数据挖掘与分析：系统方法与商业实践

作者：宋天龙

ISBN: 978-7-111-49059-3 定价：89.00元



### 社交网站的数据挖掘与分析（原书第2版）

作者：Matthew A. Russell 译者：苏统华 等

ISBN: 978-7-111-48699-2 定价：79.00元

## 内容简介

本书以算法应用为主线，由浅入深细致地讲解，从全新的角度诠释机器学习的算法理论，同时注重机器学习算法的实际运用。为了让读者便于理解，首先概述了机器学习算法，并介绍目前常用的科学计算平台和书中将用到的工程计算平台，通过应用计算平台，降低了算法实现的难度；然后，对相关计算平台应用实例进行讲解，介绍计算平台的开发基础，并讲解计算的应用与实例；最后，以数据统计分析实战和机器学习实战为重点，应用计算平台实现数据统计分析以及机器学习算法。第2版相对第1版扩充了很多内容，比如：生产环境部署、假设检验与回归、描述性分析、图像算法与机器视觉、更多的机器学习算法、自然语言处理等。此书可供IT专业人员和机器学习爱好者参考使用。

书中涉及的代码及相关资源请到<https://yunpan.cn/cYjhBYGLKkKTb>（提取码：65ad）下载，欢迎读者发送邮件到麦好的邮箱myhaspl@myhaspl.com，期待读者的真挚反馈。

## 针对本书第1版，豆瓣网友的评论

我在“陌陌”工作的时候，由于工作关系接触了本书。起初是当作工具书来阅读的，后来才发现这本书的连贯性很好，于是又重新读了一遍。本书这种从实战的角度带入问题，很快地定位问题，而且举一反三地由书中的例子解决实际工作中的问题。机器学习就是需要通过实际的例子让你的机器更“聪明”，更懂得你需要什么。强烈推荐！

—— 网友：tangmash

我从事数据挖掘工作4年，过去使用SAS进行数据挖掘，由于本职工作需要了解机器学习、Python等，而《机器学习实践指南》刚好都有所涉及。这本书真正把知识点与思路很好地串联了起来，由最开始介绍什么是机器学习，如何搭建相关环境，到算法的介绍以及代码实现，都体现作者希望读者能从实践中快速入门、融会贯通的思想。这本书起到了最核心的实践和指南作用，值得一读。期待作者在以后的作品中能带给读者更多的启发。

—— 网友：Keven\_guo

我是非计算机专业的一名大专生，本书非常适合初学者，理论不多，偏于实战。一方面能快速提高Python技能，另一方面也能快速理解机器学习。通过本书，可由浅入深、逐步解决在机器学习中遇到的疑惑。在实际开发中，本书对我帮助很大。

—— 网友：MG

本人是计算神经科学专业的在读博士，因需要快速入门机器学习，一个偶然的机会看到了《机器学习实践指南》这本书，该书帮助我快速理解了机器学习并将这些方法应用于自己的研究领域。之前看过一些其他与机器学习相关的书籍，基本上都是偏理论的，没有太多实例或者自己动手的机会。本书包含从基础理论的介绍到最后问题的解决方案和实践思路，并提供详细的源代码和数据。读者可以边看书边上机，步步为营。书中还介绍了MLPY等一些实用机器学习库，也详细介绍了像神经网络、SVM等技术方法。总之，本书很适合像我这样需要快速入门机器学习的读者。最后作者还建立了书友QQ群，让书友可以一起讨论交流，作者还会在群里与大家一起讨论和分析问题，气氛很好。

—— 网友：Boite

很偶然地看到这本书，正好自己在研究机器学习，感觉机器学习涉及的很多算法是很难懂的，然而本书让我获得了机器学习方面的很多知识并解答了许多疑惑，果断给“满分”。机器学习可以说还是一门很新的技术，它甚至可以上升到一个新学科，其中要研究的东西很多。本书侧重于实践和应用，用于机器学习启蒙是再好不过的了。

—— 网友：skyworthfly

我是深圳市芥末金融信息服务公司的一位技术员工，由于项目需求，在China-pub上看到本书。个人觉得本书是一本非常注重实践的书，比较适合初学者和想快速应用的开发人员。书中的示例比较切合实际。个人认为作者在举例方面很用心，能将难懂的机器学习原理用简单的代码快速地表达和应用。

—— 网友：john\_deng



投稿热线: (010) 88379604  
客服热线: (010) 88379426 88361066  
购书热线: (010) 68326294 88379649 68995259

华章网站: [www.hzbook.com](http://www.hzbook.com)  
网上购书: [www.china-pub.com](http://www.china-pub.com)  
数字阅读: [www.hzmedia.com.cn](http://www.hzmedia.com.cn)

上架指导: 计算机/机器学习

ISBN 978-7-111-54021-2



9 787111 540212 >

定价: 89.00元